

# Java 11 Fundamentals



**INTEC BRUSSEL**  
MEER KANSEN OP WERK

Copyright® 2019 Noël Vaes  
[www.noelvaes.eu](http://www.noelvaes.eu)

Roupplein 16  
1000 Brussel  
[www.intecbrussel.be](http://www.intecbrussel.be)

Vrijwel alle namen van software- en hardwareproducten die in deze cursus worden genoemd, zijn tegelijkertijd ook handelsmerken en dienen dienovereenkomstig te worden behandeld.

Alle rechten voorbehouden. Niets uit deze uitgave mag worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand of openbaar worden gemaakt in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnemen of op enige andere manier, zonder voorafgaande schriftelijke toestemming van de auteur. De enige uitzondering die hierop bestaat, is dat eventuele programma's en door de gebruiker te typen voorbeelden mogen worden ingevoerd opgeslagen en uitgevoerd op een computersysteem, zolang deze voor privé-doeleinden worden gebruikt, en niet bestemd zijn voor reproductie of publicatie.

Correspondentie inzake overnemen of reproductie kunt u richten aan:

Noël Vaes  
Roode Roosstraat 5  
3500 Hasselt  
België

Tel: +32 474 38 23 94

[noel@noelvaes.eu](mailto:noel@noelvaes.eu)  
[www.noelvaes.eu](http://www.noelvaes.eu)

Ondanks alle aan de samenstelling van deze tekst bestede zorg, kan de auteur geen aansprakelijkheid aanvaarden voor eventuele schade die zou kunnen voortvloeien uit enige fout, die in deze uitgave zou kunnen voorkomen.

09/01/2019

Copyright<sup>©</sup> 2019 Noël Vaes

# Inhoudsopgave

<b>Hoofdstuk 1: Inleiding.....</b>	<b>10</b>
1.1 De geschiedenis van Java.....	10
1.2 Java als programmeertaal.....	10
1.2.1 Soorten programmeertalen.....	10
1.2.2 Java versus andere programmeertalen.....	15
1.2.3 Kenmerken van Java als programmeertaal.....	16
1.3 Java als platform.....	17
1.4 Soorten Java-toepassingen.....	18
1.5 Samenvatting.....	19
<b>Hoofdstuk 2: De Java Development Kit.....</b>	<b>20</b>
2.1 Inleiding.....	20
2.2 JDK en documentatie.....	20
2.3 De omgevingsvariabele JAVA_HOME.....	22
2.4 Ontwikkelomgevingen.....	23
2.5 Samenvatting.....	28
<b>Hoofdstuk 3: Mijn eerste Java-toepassing.....</b>	<b>29</b>
3.1 Inleiding.....	29
3.2 De broncode schrijven.....	29
3.3 De broncode compileren.....	30
3.4 De bytecode uitvoeren.....	30
3.5 De opbouw van het programma.....	31
3.5.1 Commentaar in Java-code.....	31
3.5.2 Het pakket definiëren.....	32
3.5.3 De klasse definiëren.....	32
3.5.4 De methode main().....	33
3.5.5 Het eigenlijke werk.....	33
3.6 Samenvatting.....	34
<b>Hoofdstuk 4: Programmatielogica.....</b>	<b>35</b>
4.1 Inleiding.....	35
4.2 Sequenties.....	35
4.3 Invoer en uitvoer.....	37
4.4 Keuzes.....	38
4.5 Herhalingen.....	40
4.6 Samenvatting: programmeeralgoritmen.....	44
<b>Hoofdstuk 5: De Java-programmeertaal.....</b>	<b>45</b>
5.1 Inleiding.....	45
5.2 Variabelen en letterlijke waarden.....	45
5.2.1 De declaratie van variabelen.....	45
5.2.2 Het datatype .....	47
5.2.3 Literals.....	49
5.2.4 De naam.....	52
5.2.5 Final variables of constanten.....	54
5.2.6 Typeconversie.....	54
5.3 Operatoren.....	57
5.3.1 Rekenkundige operatoren.....	58
5.3.2 Relationale operatoren.....	62
5.3.3 Logische operatoren.....	63
5.3.4 Shift-operatoren.....	64



5.3.5 Bit-operatoren.....	67
5.3.6 Toekenningsoperatoren.....	72
5.3.7 Conditionele operatoren.....	73
5.3.8 Overige operatoren.....	75
5.3.9 Prioriteitsregels.....	75
5.4 Uitdrukkingen, statements en blokken.....	78
5.4.1 Uitdrukkingen.....	78
5.4.2 Statements.....	79
5.4.3 Codeblok.....	79
5.5 Programmaverloop-statements.....	80
5.5.1 Inleiding.....	80
5.5.2 Het if else statement.....	81
5.5.3 Het switch statement.....	84
5.5.4 Het while en do while statement.....	88
5.5.5 Het for statement of zelfstellende lus.....	94
5.6 Methoden.....	97
5.7 Samenvatting.....	104
<b>Hoofdstuk 6: Objectgeoriënteerd programmeren.....</b>	<b>105</b>
6.1 Inleiding.....	105
6.2 Inleiding in het objectgeoriënteerd programmeren.....	105
6.2.1 Objecten.....	105
6.2.2 Boodschappen.....	107
6.2.3 Klassen.....	109
6.3 Werken met bestaande objecten.....	110
6.3.1 Inleiding.....	110
6.3.2 Objecten maken van een bestaande klasse.....	110
6.3.3 Objecten gebruiken.....	114
6.3.4 Objecten opruimen.....	115
6.4 Tekenreeksen.....	116
6.4.1 Inleiding.....	116
6.4.2 De klasse String.....	116
6.4.3 De klasse StringBuilder.....	126
6.4.4 Strings samenvoegen met de + operator.....	128
6.4.5 Gegevens formatteren met de klasse Formatter.....	129
6.5 Samenvatting.....	134
<b>Hoofdstuk 7: Arrays.....</b>	<b>135</b>
7.1 Inleiding.....	135
7.2 Arrays maken.....	135
7.3 Arrays gebruiken.....	137
7.4 De uitgebreide for-lus (for each).....	138
7.5 Arrays van objecten.....	139
7.6 Arrays van arrays.....	141
7.7 Lookup tables.....	144
7.8 Methoden met een variabel aantal parameters.....	145
7.9 Samenvatting.....	146
<b>Hoofdstuk 8: Klassen definiëren.....</b>	<b>148</b>
8.1 Inleiding.....	148
8.2 De declaratie van de klasse.....	149
8.3 De klassenomschrijving (body).....	150
8.3.1 Eigenschappen.....	151
8.3.2 Methoden.....	153
8.3.3 Constructors.....	162
8.3.4 Instance members en class members.....	165
8.3.5 De klasse Math.....	171

8.4 Samenvatting.....	173
<b>Hoofdstuk 9: Associaties.....</b>	<b>174</b>
9.1 Inleiding.....	174
9.2 Associaties.....	174
9.3 Aggregaties.....	175
9.4 Composities.....	177
9.5 High cohesion.....	178
9.6 Samenvatting.....	178
<b>Hoofdstuk 10: Overerving en klassenhiërarchie.....</b>	<b>180</b>
10.1 Inleiding.....	180
10.1.1 Subklassen en superklassen.....	180
10.1.2 Overerving.....	180
10.1.3 Klassenhiërarchie.....	181
10.1.4 Abstracte klassen.....	182
10.2 Subklassen definiëren in Java.....	182
10.3 Eigenschappen van subklassen.....	183
10.3.1 Overerven van eigenschappen.....	183
10.3.2 Toevoegen van eigenschappen.....	184
10.3.3 Vervangen (verbergen) van eigenschappen.....	184
10.4 Methoden van subklassen.....	185
10.4.1 Overerven van methoden.....	185
10.4.2 Toevoegen van methoden.....	186
10.4.3 Vervangen van methoden (override).....	187
10.4.4 Polymorfisme.....	189
10.5 Constructors van subklassen.....	191
10.6 Klasseneigenschappen en klassenmethoden.....	192
10.7 Final-klassen en methoden.....	194
10.8 Abstracte klassen.....	195
10.9 De superklasse Object.....	197
10.9.1 Klassenhiërarchie.....	197
10.9.2 De operator instanceof.....	198
10.9.3 Methoden van de Object-klasse.....	199
10.10 Polymorfisme (bis).....	202
10.11 Code hergebruik: overerving versus associaties.....	203
10.12 Samenvatting.....	206
<b>Hoofdstuk 11: De opsomming.....</b>	<b>207</b>
11.1 Inleiding.....	207
11.2 Eigenschappen, methoden en constructors.....	209
11.3 Samenvatting.....	211
<b>Hoofdstuk 12: Eenvoudige klassen.....</b>	<b>212</b>
12.1 Inleiding.....	212
12.2 Wrappers voor primitieve datatypes.....	212
12.2.1 Wrapper-klassen.....	212
12.2.2 Autoboxing.....	213
12.2.3 Static members.....	216
12.3 Datums en tijden.....	217
12.3.1 Inleiding.....	217
12.3.2 Computertijden: de klasse Instant.....	218
12.3.3 Menselijke datums en tijden.....	220
12.3.4 Tijdsduur.....	223
12.3.5 Formatting van datums en tijden.....	224
12.3.6 Omzetting van en naar Date en Calendar.....	226
12.4 Samenvatting.....	226

<b>Hoofdstuk 13: Interfaces.....</b>	<b>227</b>
13.1 Inleiding.....	227
13.2 Een interface definiëren.....	228
13.2.1 De declaratie van de interface.....	228
13.2.2 De beschrijving van de interface.....	229
13.3 Een interface implementeren in een klasse.....	231
13.4 Standaardmethoden.....	232
13.5 Statische methoden.....	233
13.6 De interface als datatype.....	234
13.7 Samenvatting.....	237
<b>Hoofdstuk 14: Geneste en anonieme klassen.....</b>	<b>238</b>
14.1 Inleiding.....	238
14.2 Gewone geneste klassen (inner classes).....	238
14.3 Lokale geneste klassen (local inner classes).....	240
14.4 Anonieme geneste klassen (anonymous inner classes).....	241
14.5 Static geneste klassen (static nested classes).....	242
14.6 Samenvatting.....	245
<b>Hoofdstuk 15: Exception handling.....</b>	<b>246</b>
15.1 Inleiding.....	246
15.2 Exceptions afhandelen.....	246
15.2.1 Een exception veroorzaken.....	247
15.2.2 Een exception opvangen.....	248
15.2.3 Meerdere exceptions opvangen.....	250
15.2.4 Gemeenschappelijke exception handlers.....	251
15.2.5 Het finally blok.....	253
15.3 Exceptions genereren.....	255
15.3.1 Het throw-statement.....	255
15.3.2 Exceptions bij vervangen methoden.....	257
15.4 Soorten exceptions.....	257
15.4.1 Exceptions versus errors.....	257
15.4.2 Checked exceptions versus runtime exceptions.....	258
15.5 Zelf een exception-klasse maken.....	259
15.6 Exceptions opvangen, inpakken en verder gooien.....	260
15.7 Samenvatting.....	262
<b>Hoofdstuk 16: Javadoc.....</b>	<b>263</b>
16.1 Inleiding.....	263
16.2 Javadoc tags.....	263
16.2.1 Documentatie van klassen en interfaces.....	263
16.2.2 Documentatie van eigenschappen.....	265
16.2.3 Documentatie van methoden en constructors.....	265
16.2.4 Documentatie van pakketten.....	266
16.2.5 Overzichtsdocumentatie.....	266
16.3 JAVADOC-tool.....	266
16.4 Samenvatting.....	267
<b>Hoofdstuk 17: Generieken.....</b>	<b>268</b>
17.1 Inleiding.....	268
17.2 Generieke klassen.....	268
17.2.1 Generieken definiëren.....	269
17.2.2 Het gebruikte type inperken.....	274
17.2.3 Onbepaald type.....	276
17.2.4 Subklassen van generieke klassen.....	277
17.3 Generieke interfaces.....	277
17.4 Generieke methoden.....	281

17.4.1 Formele generieke parameters.....	281
17.4.2 Formele generieke parameters met wildcards.....	281
17.4.3 Formele generieke parameters met bounded wildcards.....	282
17.4.4 Type-parameters.....	284
17.5 Achter de schermen van de generieken.....	285
17.6 Arrays en generieken.....	286
17.7 Samenwerking tussen oude en nieuwe code.....	287
17.8 Samenvatting.....	288
<b>Hoofdstuk 18: Lambda Expressions.....</b>	<b>289</b>
18.1 Inleiding.....	289
18.2 Functionele interfaces.....	291
18.3 Definitie van lambda expressions.....	291
18.4 Methodereferenties.....	294
18.4.1 Statische methoden van een klasse of interface.....	294
18.4.2 Methoden van een gebonden object.....	296
18.4.3 Methoden van een ongebonden object.....	297
18.4.4 Constructorreferenties.....	297
18.5 Standaard functionele interfaces.....	300
18.5.1 Predicate<T>.....	300
18.5.2 Function<T,R>.....	301
18.5.3 Consumer<T>.....	303
<b>Hoofdstuk 19: Streaming API.....</b>	<b>304</b>
19.1 Inleiding: interne versus externe iteraties.....	304
19.2 Bron van streams.....	305
19.3 Bewerkingen.....	307
19.3.1 Eindbewerkingen.....	307
19.3.2 Tussenliggende bewerkingen.....	311
19.4 Samenvatting.....	315
<b>Hoofdstuk 20: Collections.....</b>	<b>316</b>
20.1 Het Collections Framework.....	316
20.2 De interface Collection en implementaties.....	316
20.2.1 List.....	319
20.2.2 Set.....	325
20.2.3 SortedSet & NavigableSet.....	330
20.2.4 Queue.....	332
20.2.5 Deque.....	334
20.2.6 Vergelijking tussen de implementaties.....	336
20.2.7 Het sorteren van verzamelingen.....	336
20.2.8 Collections en streams.....	344
20.3 De interface Map en implementaties.....	345
20.3.1 Map.....	346
20.3.2 SortedMap & NavigableMap.....	349
20.3.3 Vergelijking tussen de implementaties.....	350
<b>Hoofdstuk 21: Lezen en schrijven (I/O).....</b>	<b>351</b>
21.1 Inleiding.....	351
21.2 Mappen en bestanden.....	351
21.2.1 De interface Path.....	351
21.2.2 De klasse FileSystem.....	354
21.2.3 De klasse Files.....	354
21.2.4 De klasse File.....	357
21.3 IO-streams.....	357
21.3.1 Character streams.....	359
21.3.2 Byte streams.....	367
21.4 Object Serialization.....	372



21.4.1 Objecten serialiseren en deserialiseren.....	372
21.4.2 Klassen serialiseerbaar maken.....	373
21.4.3 Transiënte variabelen.....	375
21.4.4 Het serialisatiemechanisme aanpassen.....	377
21.4.5 Serialisatie en overerving.....	378
21.4.6 Versienummering.....	378
21.5 Programma-attributen.....	380
<b>Hoofdstuk 22: Java via de commandolijn.....</b>	<b>383</b>
22.1 Inleiding.....	383
22.2 Compileren.....	383
22.3 Modules maken.....	387
22.3.1 Inleiding.....	387
22.3.2 Een module definiëren.....	388
22.3.3 Pakketten exporteren.....	388
22.3.4 Afhankelijkheden van andere modules.....	389
22.3.5 Transitieve afhankelijkheden.....	392
22.3.6 Automatische modules.....	393
22.4 JAR-bestanden maken.....	393
22.4.1 Basisprincipes van een JAR.....	394
22.4.2 Een JAR-bestand maken.....	394
22.4.3 Een JAR-bestand opnemen in het modulepad.....	396
22.4.4 Resources uit een JAR-bestand lezen.....	399
22.5 Programma's uitvoeren.....	401
22.6 Automatische modules.....	402
22.7 Linken.....	402
<b>Hoofdstuk 23: Systeembronnen gebruiken.....</b>	<b>405</b>
23.1 Inleiding.....	405
23.2 De System-klasse.....	405
23.2.1 Standaard-I/O streams.....	405
23.2.2 Systeemeigenschappen.....	410
23.2.3 Overige methoden.....	412
23.3 Het Runtime object.....	412
23.4 De ProcessBuilder.....	413
<b>Hoofdstuk 24: Multithreading.....</b>	<b>414</b>
24.1 Inleiding: multiprocessing en multithreading.....	414
24.2 Een nieuwe thread creëren.....	415
24.2.1 Subklasse van de klasse Thread.....	416
24.2.2 De interface Runnable.....	418
24.2.3 Thread met lambda expression.....	419
24.3 De levenscyclus van threads.....	420
24.4 De uitvoering van threads in de toestand RUNNABLE.....	421
24.4.1 De scheduler.....	421
24.4.2 Prioriteiten van threads.....	422
24.4.3 Preëmptieve multitasking.....	423
24.4.4 Coöperatieve multitasking.....	423
24.5 Daemon threads.....	424
24.6 De wachttoestand.....	425
24.6.1 De slaaptoestand.....	426
24.6.2 Wachten op de beëindiging van een andere thread.....	428
24.7 Synchronisatie van threads (monitoring).....	429
24.7.1 Object locking.....	430
24.7.2 Wait() en notify().....	434
24.8 De Timer-klasse en de TimerTask-klasse.....	437
24.9 Concurrency framework.....	438

24.9.1 Concurrent collections.....	438
24.9.2 Atomaire objecten.....	440
24.9.3 Callable, ExecutorService and Future.....	442
24.10 Parallelisme met streams.....	444

## **Hoofdstuk 25: Grafische applicaties met JavaFX.....446**

25.1 Inleiding.....	446
25.2 Installatie van JavaFX.....	447
25.3 Mijn eerste JavaFX-toepassing.....	449
25.4 FXML.....	450
25.5 Stage, Scenes en Nodes.....	452
25.6 Model View Controller.....	455
25.7 Controls.....	460
25.7.1 Label.....	461
25.7.2 TextField en PasswordField.....	462
25.7.3 Button.....	463
25.7.4 CheckBox.....	463
25.7.5 RadioButton.....	464
25.7.6 ChoiceBox.....	465
25.7.7 ComboBox.....	466
25.7.8 ListView.....	467
25.7.9 Slider.....	469
25.7.10 ScrollBar.....	470
25.7.11 DatePicker.....	470
25.7.12 ColorPicker.....	471
25.8 Layout met Panes.....	471
25.8.1 BorderPane.....	471
25.8.2 HBox.....	472
25.8.3 VBox.....	473
25.8.4 GridPane.....	473
25.8.5 FlowPane.....	474
25.8.6 TilePane.....	475
25.8.7 AnchorPane.....	476
25.8.8 ScrollPane.....	476
25.8.9 Combinatie van panes.....	477
25.9 Menu's.....	479
25.10 Event Handling.....	481
25.11 Tekenen met Canvas.....	485
25.12 Cascading Style Sheets (CSS).....	489
25.12.1 CSS-bestanden.....	489
25.12.2 Selectors.....	490
25.12.3 Stijlkenmerken via code instellen.....	493
25.13 Dialoogvensters.....	495
25.13.1 Alert.....	495
25.13.2 TextInputDialog.....	497
25.13.3 FileChooser.....	497
25.14 Samenvatting.....	499

## Hoofdstuk 1: Inleiding

### 1.1 De geschiedenis van Java

De programmeertaal Java werd in 1995 ontwikkeld door het bedrijf SUN. Aanvankelijk waren Java en de voorganger OAK bedoeld als robuuste programmeertaal voor consumentenelektronica. Men wou namelijk een taal die betrouwbaar was, die objectgeoriënteerd was en die onafhankelijk was van de snel evoluerende computerchips.

Met de opkomst van het internet stelde men vast dat Java uitermate geschikt was voor een dergelijk groot netwerk dat bestaat uit heterogene computersystemen. Door zijn platformonafhankelijk karakter kunnen de programma's namelijk overal ingezet worden.

Intussen is Java uitgegroeid tot een programmeertaal en platform en is niet meer weg te denken uit het firmament van de softwareontwikkeling. Java wordt momenteel gebruikt voor het bouwen van platformonafhankelijke desktopapplicaties maar vooral voor het maken van *enterprise-applicaties* (*multitier* gedistribueerde applicaties). Dynamische webapplicaties maken daar een deel van uit.

Java is zowel een **programmeertaal** als een **platform**. Eerst beschrijven we de kenmerken van Java als programmeertaal en vervolgens haar eigenschappen als platform.

### 1.2 Java als programmeertaal

Java is zowat een buitenbeentje tussen de overige programmeertalen. Java weet de voordelen van verschillende soorten programmeertalen in zich te verenigen.

We zullen eerst trachten Java te situeren tussen de andere programmeertalen.

#### 1.2.1 Soorten programmeertalen

Een computer kan slechts werken met binaire codes. Iedere instructie die hij uitvoert, is eigenlijk een binair getal dat opgeslagen is in het werkgeheugen. De processor haalt dit getal (instructie) uit het geheugen en voert de instructie uit. Deze binaire codes en de overeenkomstige instructies zijn specifiek voor iedere processor of processorfamilie. Zo heeft een processor van Intel een andere instructieset dan de SPARC van SUN. Beide zijn op binair niveau helemaal niet compatibel. Binaire codes voor de Intel kunnen niet door de SPARC gebruikt worden en omgekeerd.

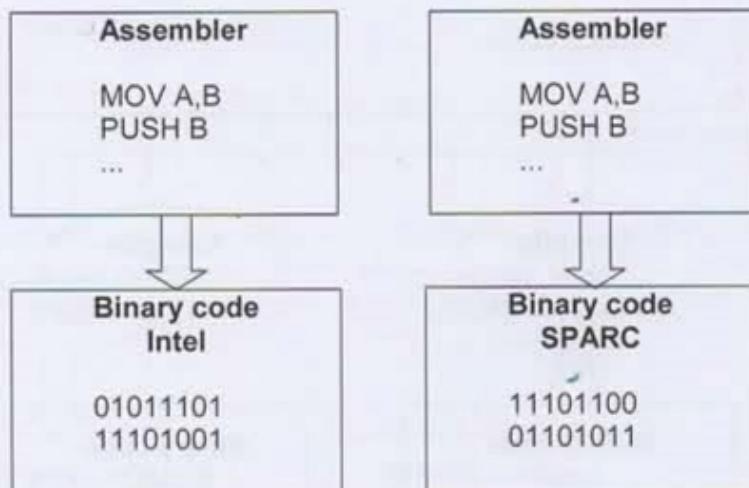
Binary code Intel	Binary code SPARC
01100110	01011001
11010001	01010111
...	...

Afbeelding 1: Binaire code Intel versus SPARC

De allereerste programmeurs schreven programma's rechtstreeks in **binaire code**, ook wel machinetaal genoemd. Dit programmeerwerk was vrij omslachtig en tijdrovend. Deze binaire codes zijn niet gebruiksvriendelijk en de kans op het maken van fouten is zeer groot. Machinetaal wordt ook wel de "**eerste generatie programmeertaal**" genoemd.

Om deze vorm van programmeren makkelijker te maken, werd de programmeertaal **Assembler** ontwikkeld. Dit is een "**tweede generatie programmeertaal**". Bij Assembler

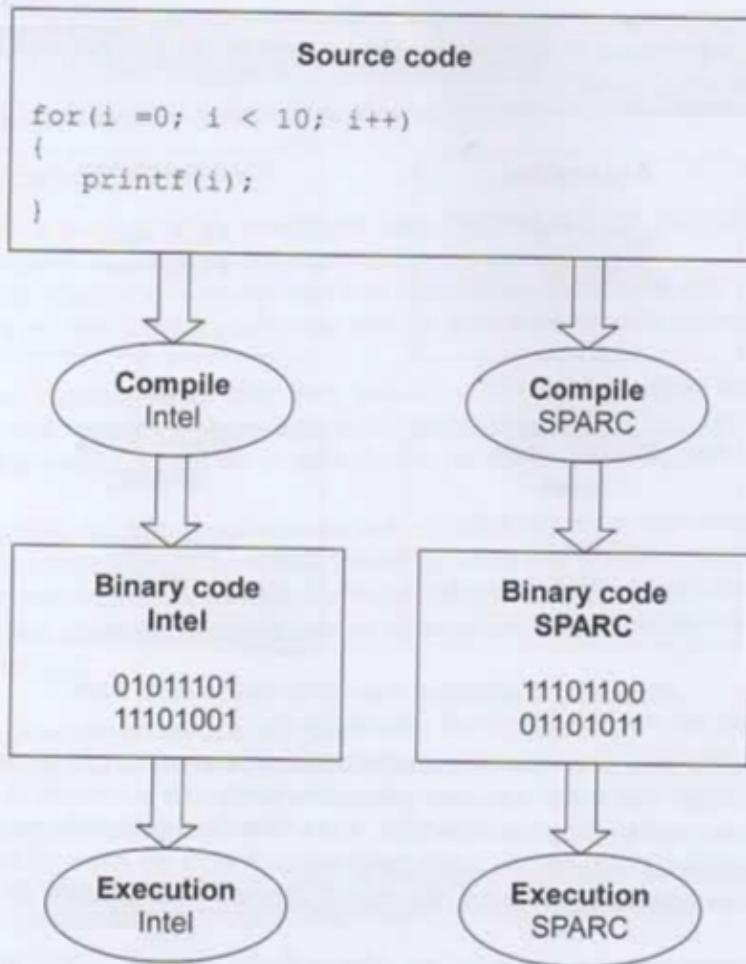
worden de binaire codes vervangen door gebruiksvriendelijker woorden en symbolen. Het programma wordt geschreven in deze *Assembler*-codes en nadien vertaald in de overeenkomstige binaire codes.



Afbeelding 2: Vertaling assembler naar binaire code

De *Assembler*-programmacode voor de verschillende processoren lijkt al meer op elkaar, maar toch is *Assembler* niet meer dan een gebruiksvriendelijke voorstelling van de binaire code. Het is dus geen echte programmeertaal. *Assembler* maakt het de programmeur gewoon wat makkelijker. Ondanks de grote gelijkenissen blijft de *Assembler*-taal toch specifiek voor iedere processor en is ze niet overdraagbaar naar andere processoren.

Bij hogere programmeertalen, zoals C/C++, Visual Basic, Pascal, Cobol enzovoort wordt de programmacode geschreven in een vrij gebruiksvriendelijke taal: met woorden in plaats van met binaire codes. Men noemt dit de 'broncode'. Zo'n programma wordt nadien omgezet in de juiste binaire code voor een bepaalde processor. Dit noemt men de 'objectcode'. Deze programmeertalen noemt men ook wel "derde generatie programmeertalen".



Afbeelding 3: Compilatie van broncode

Sommige hogere programmeertalen (zoals C/C++) zijn overdraagbaar. Dat wil zeggen dat een programma geschreven in die taal onafhankelijk is van het type processor dat nadien de instructies zal uitvoeren. De programmacode wordt nadien vertaald naar de juiste binaire instructies voor die specifieke processor.

Het omzetten van die programmaregels naar die binaire code kan op twee verschillende momenten gebeuren: ofwel op voorhand ofwel tijdens de uitvoering van het programma.

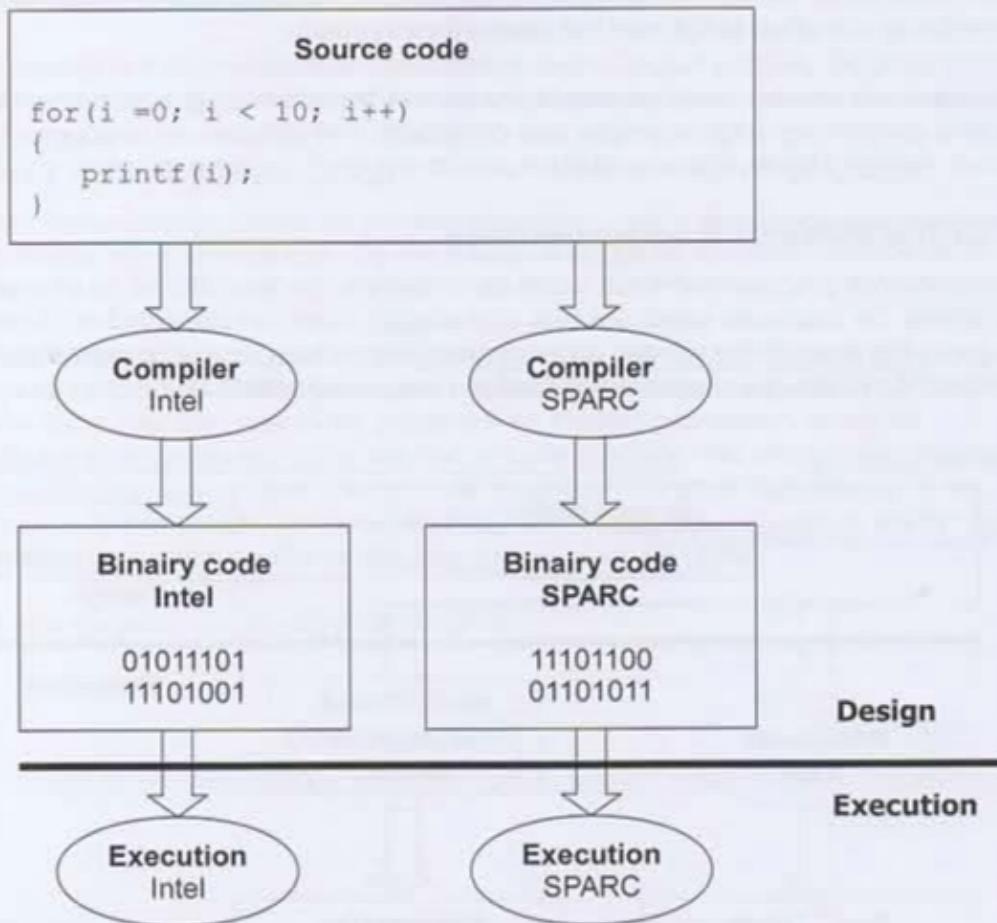
Op basis van dit vertaalmoment worden programmeertalen in twee groepen verdeeld:

1. Gecompileerde programmeertalen
2. Geïnterpreteerde programmeertalen

### 1.2.1.1 Gecompileerde programmeertalen

Bij gecompileerde programmeertalen wordt de broncode weggeschreven in een tekstbestand. Deze broncode wordt vervolgens vertaald naar de binaire objectcode die wordt weggeschreven in een uitvoerbaar binair bestand. Men noemt dit proces 'compilieren' en dit wordt gedaan door een *compiler*.

Nadien wordt de binaire code van het bestand ingeladen en uitgevoerd door de processor.



Afbeelding 4: Gecompileerde programmeertalen

Ieder type processor heeft zijn eigen compiler die de programmacode kan omzetten in de juiste binaire codes voor de processor.

#### Voordelen:

1. De broncode van gecompileerde talen is overdraagbaar. Men kan programma's schrijven in één taal en toch laten uitvoeren op verschillende machines.
2. Gecompileerde programma's zijn **snel** omdat de binaire code rechtstreeks wordt uitgevoerd.
3. De objectcode is binair en kan dus moeilijk aangepast of gebruikt worden door anderen. Zonder de overeenkomstige broncode is het haast onmogelijk te achterhalen hoe een programma is opgebouwd. De broncode is dus goed beschermd.

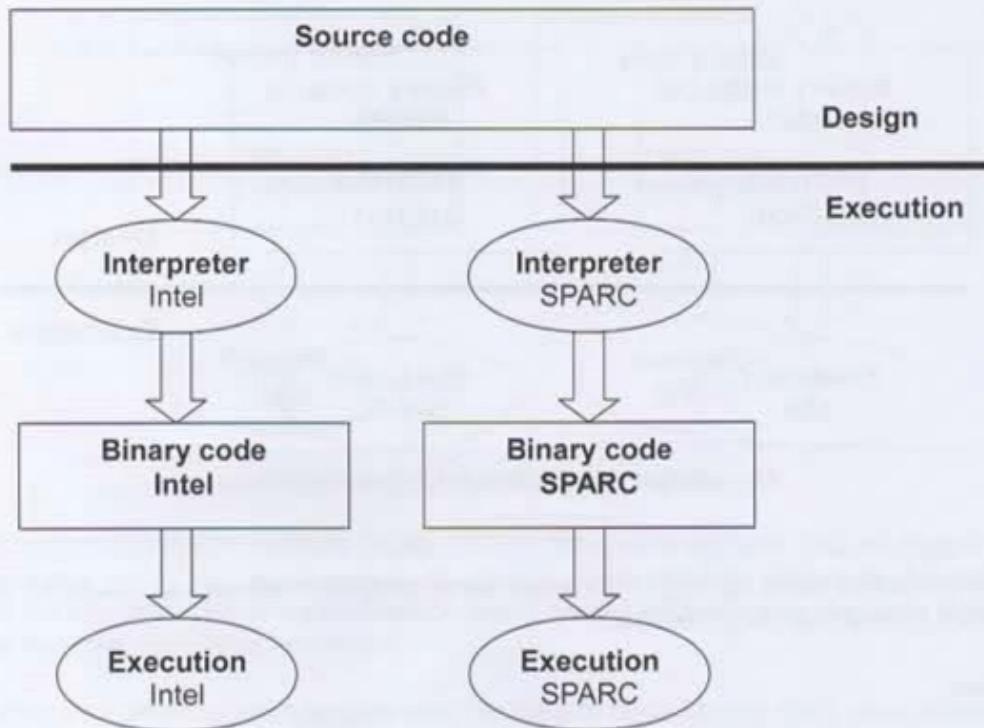
#### Nadelen:

1. Voor elk type processor moet een afzonderlijk binair bestand (objectcode) gemaakt worden. De uitvoerbare programma's zijn niet overdraagbaar. De objectcode is met andere woorden **processorafhankelijk**. Dit vormt een probleem als programma's bijvoorbeeld over het internet verspreid worden. Er moet dan voor elk type computer een afzonderlijk uitvoerbaar bestand gemaakt worden.

2. Voor elk besturingssysteem moet het programma afzonderlijk gecompileerd worden omdat de interactie met het besturingssysteem telkens anders is. Zowel de broncode als de objectcode zijn **afhankelijk van het besturingssysteem**.
3. De programma's moeten eerst gecompileerd worden vooraleer ze getest kunnen worden. Na iedere aanpassing volgt nogmaals een compilatie. Het uittesten en *debuggen* is daardoor **omslachtig en tijdrovend**.

### 1.2.1.2 Geïnterpreteerde programmeertalen

Bij geïnterpreteerde programmeertalen wordt de vertaalslag gedaan tijdens de uitvoering van het programma. De broncode wordt ook hier opgeslagen in een tekstbestand en tijdens de uitvoering van het programma worden de programmaregels stap voor stap geïnterpreteerd en uitgevoerd. Er is dus geen intermediair bestand met objectcode.



Afbeelding 5: Geïnterpreteerde programmeertalen

Het interpreteren wordt in dit geval gedaan door een *interpreter*. Scripttalen (zoals *JavaScript*, *Visual Basic Script*) zijn over het algemeen geïnterpreteerde talen. In dit geval is het bijvoorbeeld de internetbrowser die dienst doet als *interpreter*.

#### Voordelen:

1. De programmacode kan snel aangepast worden en onmiddellijk geëvalueerd worden.
2. Programma's zijn onmiddellijk overdraagbaar, omdat de programmacode onafhankelijk is van de processor en het besturingssysteem. De vertaling gebeurt namelijk door de *interpreter*. Dit maakt dit soort talen uitermate geschikt voor verspreiding via het internet. Er is slechts één broncode die rechtstreeks kan dienen voor verschillende platformen.

#### Nadelen:

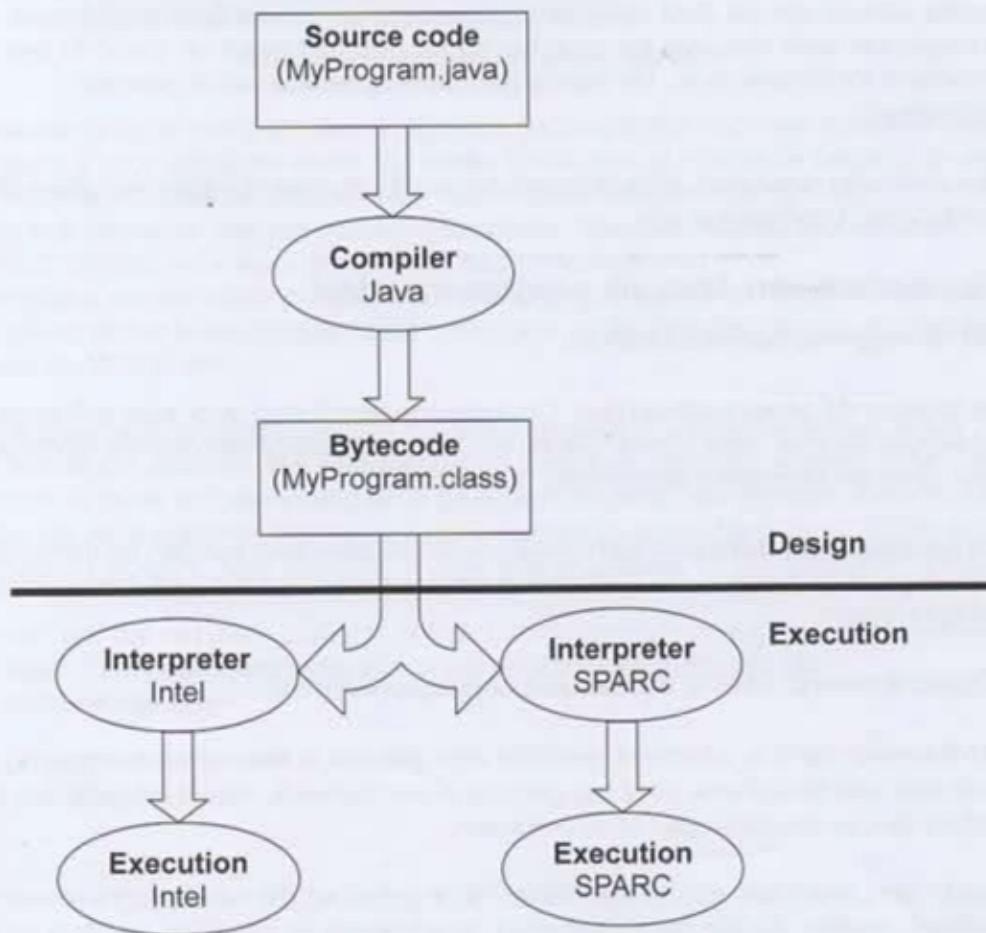
1. De programma's werken traag, omdat alle programmastappen telkens weer geïnterpreteerd moeten worden.
2. Het is moeilijk om de broncode te beschermen tegen illegaal gebruik. De programma's bestaan uit tekstbestanden die anderen naar believen kunnen kopiëren en aanpassen.

### 1.2.2 Java versus andere programmeertalen

Java is een buitenbeentje tussen de programmeertalen. Het is zowel een gecompileerde als geïnterpreteerde programmeertaal. Op die manier weet ze de voordelen van beide in zich te verenigen.

Een Java-programma wordt geschreven in een gewoon tekstbestand (**broncode**) met extensie **java** (voorbeeld **MyProgram.java**). In plaats van deze broncode te vertalen naar een binaire code voor een specifieke processor en besturingssysteem, wordt hij **gecompileerd** naar de binaire code van een virtuele machine met een **virtuele processor** en **virtueel besturingssysteem**. Men noemt dit de '**bytecode**'. Hij wordt opgeslagen in een bestand met extensie **class** (voorbeeld **MyProgram.class**). Deze **bytecode** wordt nadien **geïnterpreteerd** en uitgevoerd door de **Java Virtual Machine (JVM)**.

Dit wordt weergegeven in het volgende schema:



Afbeelding 6: Java als gecompileerde en geïnterpreteerde programmeertaal

#### Voordelen:

1. Gecompileerde Java-programma's zijn **overdraagbaar**. De **bytecode** is universeel en kan

door elke JVM gebruikt worden. Dit maakt Java uitermate geschikt voor het gebruik op het internet.

2. Vanwege van de compacte en efficiënte *bytecode* is Java snelter dan de meeste geïnterpreteerde talen.
3. De *bytecode* kan bovendien ook nog gecomprimeerd worden en voorzien worden van een digitale handtekening. Dit is vooral interessant als software wordt gedownload van het internet.
4. De *bytecode* is beter beschermd tegen illegaal gebruik en aanpassingen.
5. Java is niet enkel processoronafhankelijk maar ook platformonafhankelijk.

#### Nadelen:

1. Java is **trager** dan pure gecompileerde programmeertalen omdat de *bytecode* uiteindelijk toch geïnterpreteerd moet worden. Dit euvel tracht men op te lossen door gebruik te maken van een *JIT compiler (Just In Time compiler)*. Deze compileert de Java-*bytecode* in binaire code de eerste keer dat de code uitgevoerd wordt. Het programma wordt dus net op tijd (*just in time*) gecompileerd. Dit zorgt aanvankelijk voor de nodige vertraging. De laatste versies van de JVM zijn echter gebaseerd op de *HotSpot*-technologie. Hierbij wordt nagegaan welk deel van de code het meest gebruikt wordt en enkel dit deel wordt gecompileerd tot binaire code. De weinig gebruikte *bytecode* wordt gewoon geïnterpreteerd.
2. Op elke computer waar een Java-programma wordt uitgevoerd, moet een *Java Virtual Machine (JVM)* beschikbaar zijn.

### 1.2.3 Kenmerken van Java als programmeertaal

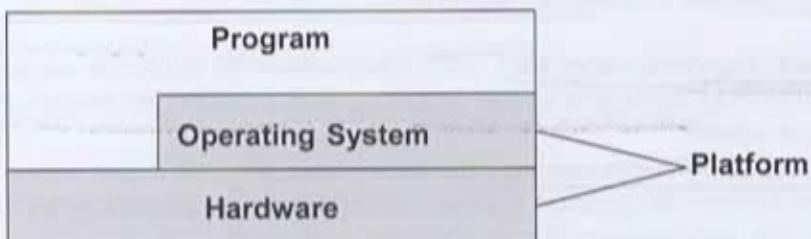
Java heeft de volgende hoofdkenmerken:

1. **Geïnterpreteerde programmeertaal:** De *bytecode* wordt stap voor stap geïnterpreteerd en uitgevoerd door de *Java Virtual Machine*. Door de *Hotspot*-technologie wordt de kritische code gecompileerd naargelang het nodig is.
2. **Overdraagbaar – platformonafhankelijk:** Java-toepassingen kunnen op verschillende platformen gebruikt worden. De *bytecode* is onafhankelijk van het type processor en het besturingssysteem.
3. **Objectgeoriënteerd:** Java is consequent objectgeoriëenteerd.
4. **Gedistribueerd:** Java is uitermate geschikt voor gebruik in een netwerkomgeving. Java is uitgerust met een bibliotheek voor het gebruik in een netwerk. Het is mogelijk om met Java *client-server*-toepassingen te ontwikkelen.
5. **Robuust:** Java heeft een aantal mechanismen ingebouwd die deze programmeertaal zeer robuust maken. Zo zijn datatypes strikt gedefinieerd, er zijn geen *pointers* en voor het geheugenbeheer wordt gebruikgemaakt van *garbage collection* waardoor vervelende *memory leaks* vermeden worden.
6. **Multithreaded:** Java biedt de mogelijkheid programma's te schrijven met meerdere uitvoeringsaders (*threads*). Hierdoor kunnen in een Java-toepassing meerdere taken tegelijkertijd uitgevoerd worden.

7. **Veilig:** Java heeft een aantal mechanismen die de veiligheid van de toepassing waarborgen.
8. **Snel:** Hoewel Java als geïnterpreteerde taal aanzienlijk trager is dan pure gecompileerde talen, kan door middel van de **HotSpot**-technologie de uitvoeringssnelheid van gecompileerde talen toch benaderd worden.

### 1.3 Java als platform

Onder platform verstaan we de combinatie van hardware en een besturingssysteem. Het meest bekende platform is het **WIntel**-platform. **WIntel** is een samenvoeging van **Windows** en **Intel**. Windows is het besturingssysteem dat gebruik maakt van de hardware op basis van Intel-processoren (of compatibele processoren).



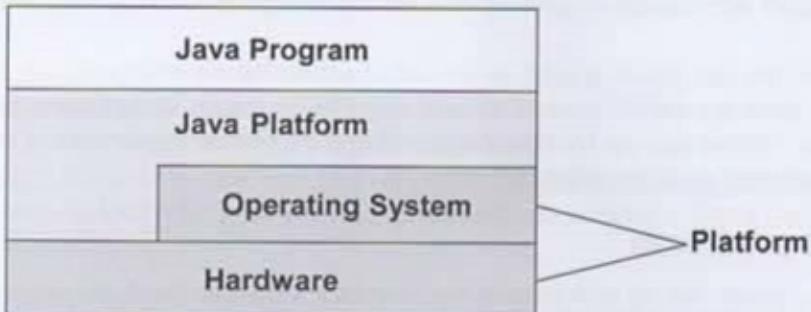
Afbeelding 7: Beteenis van een platform

Gecompileerde programma's worden doorgaans gecompileerd voor een specifiek platform. Een programma voor *Windows* werkt niet onder *Linux*, ook al maken ze beide gebruik van dezelfde hardware. Naast de juiste binaire instructies die afhankelijk zijn van de hardware, is er namelijk ook interactie met het besturingssysteem. Daarom moeten programma's opnieuw gecompileerd worden voor ieder afzonderlijk besturingssysteem.

Na de compilatie worden deze programma's namelijk gekoppeld aan bibliotheken die de communicatie met het besturingssysteem verzorgen. In de *Windows*-omgeving hebben we bijvoorbeeld de *WIN32-API*.

Java is niet enkel een programmeertaal zoals beschreven in vorige paragraaf, maar Java biedt ook een eigen platform aan waarbinnen de Java-toepassingen worden uitgevoerd. Het Java-platform is louter softwarematig en is gebouwd bovenop het gewone platform. Dit zeggen dat het Java-platform abstractie maakt van het concrete hardwareplatform en de programmacode isoleert. Juist hierdoor is Java overdraagbaar en platformonafhankelijk.

Dit impliceert wel dat het Java-platform zelf niet platformonafhankelijk is. Ieder platform moet over zijn eigen JVM beschikken. Het zijn enkel de Java-programma's die platformonafhankelijk zijn.

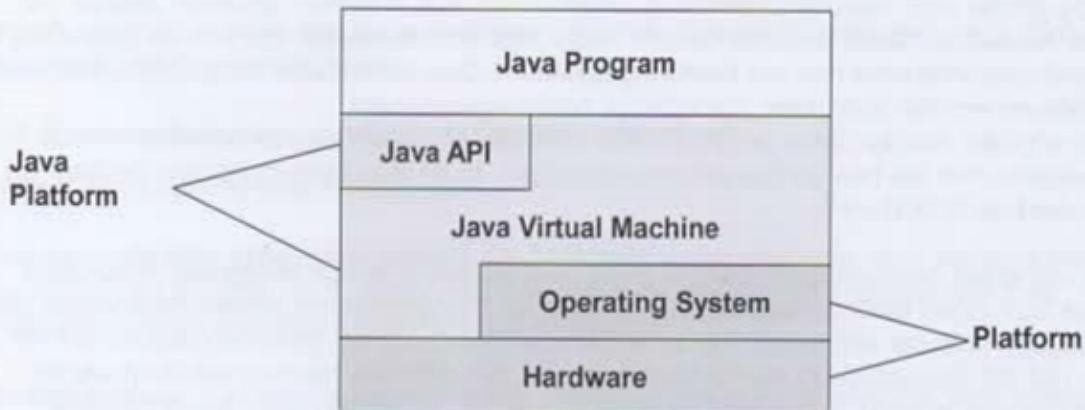


Afbeelding 8: Java als platform

Het Java-platform bestaat uit twee componenten:

1. De **Java Virtual Machine (Java VM)**: deze interpreteert de *bytecode* en maakt gebruik van de onderliggende hardware en het onderliggende besturingssysteem om de instructies uit te voeren.
2. De **Java Application Programming Interface (Java API)**: dit is een verzameling van softwarecomponenten die gebruikt kunnen worden door het Java-programma. Deze componenten zijn gegroepeerd in zogenaamde *packages*.

Het complete schema ziet er dan als volgt uit:



Afbeelding 9: Onderdelen van het Java-platform

## 1.4 Soorten Java-toepassingen

Java-toepassingen bestaan in verschillende vormen:

1. **Java-desktopapplicaties**: Dit zijn *standalone*-toepassingen die net als andere programma's worden uitgevoerd op de computer. De JVM op de computer interpreteert de *bytecode* en voert de instructies uit. Om Java-toepassingen uit te voeren moet men eerst de JVM installeren op de computer.
2. **Java-serverapplicaties**: Dit zijn Java-applicaties die uitgevoerd worden op een (web)server. Doorgaans zijn deze toepassingen toegankelijk via de webbrowser. Het is in dit geval niet nodig de JVM te installeren op de computer aangezien alle code wordt uitgevoerd op de server.

## 1.5 Samenvatting

In dit hoofdstuk hebben we gezien dat er verschillende soorten programmeertalen zijn: de gecompileerde talen en de geïnterpreteerde talen. Beide hebben hun voordelen en nadelen. Java is zowel een gecompileerde als geïnterpreteerde taal waardoor de voordelen van beide gecombineerd worden. Daarnaast is Java meer dan een programmeertaal; het is ook een eigen platform dat abstractie maakt van het onderliggende concrete platform. Hierdoor zijn Java-toepassingen echt platformonafhankelijk.

## Hoofdstuk 2: De Java Development Kit

### 2.1 Inleiding

In dit hoofdstuk leren we wat een ontwikkelaar nodig heeft om Java-toepassingen te ontwikkelen. Tevens zullen we deze benodigdheden installeren op ons systeem.

### 2.2 JDK en documentatie

Om Java-programma's te kunnen ontwikkelen en uitvoeren hebben we allerlei toepassingen nodig om code te compileren, te debuggen en uit te voeren via de *Java Virtual Machine (JVM)*. Deze toepassingen zijn samengebracht in de **Java Development Kit (JDK)**.

Op basis van deze JDK kan een aangepaste **Java Runtime Environment (JRE)** gemaakt worden die enkel bedoeld is voor het uitvoeren van de code en die samen met de programmacode geïnstalleerd kan worden op het systeem waar de toepassing zal moeten draaien. We zullen later zien hoe we zo'n JRE kunnen aanmaken.

Daarnaast moeten we ook beschikken over de nodige **documentatie**: deze kunnen we raadplegen op het internet of lokaal op ons systeem installeren.

Samengevat hebben we dus het volgende nodig:

- De **Java Development Kit (JDK)**.
- De **Java-API-documentatie**.

De JDK en de bijbehorende documentatie kunnen gratis van het internet gehaald worden op de volgende website: <http://java.oracle.com>

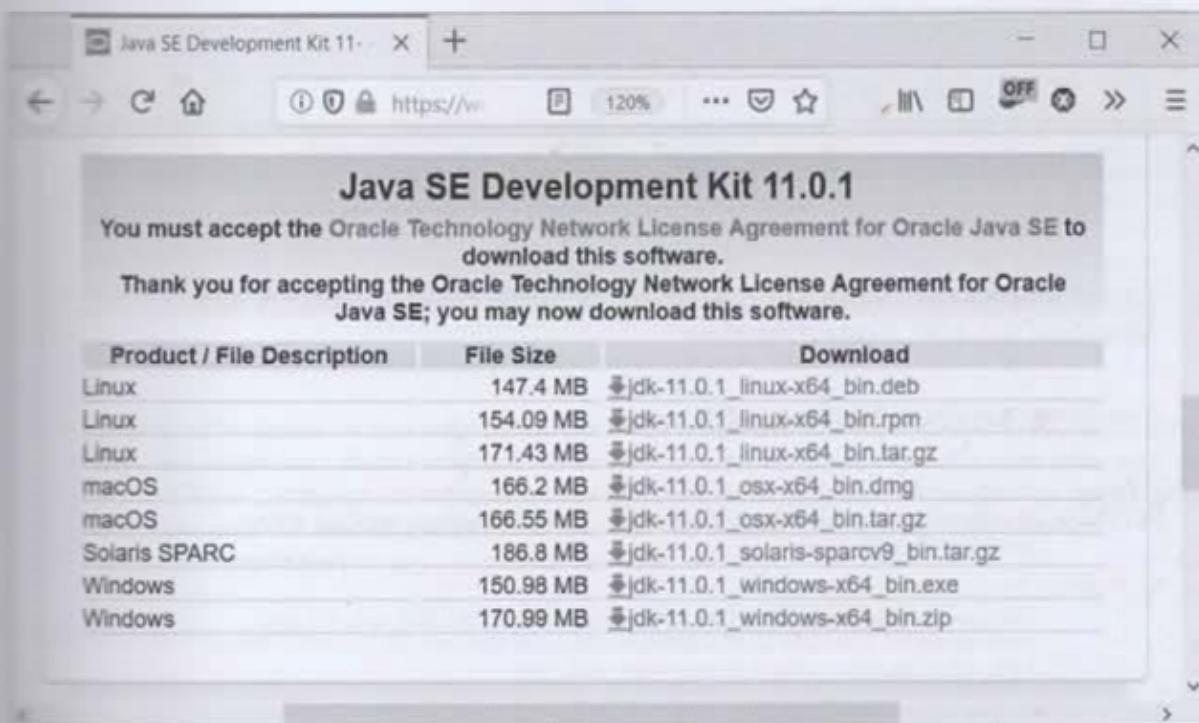
Aangezien de JVM zelf en de bijbehorende hulpprogramma's niet platformonafhankelijk zijn, dient men de juiste versie van de JDK te downloaden.

#### Opdracht 1: De JDK installeren

In deze opdracht gaan we JDK 11 van de website plukken en installeren. Tevens zullen we de omgevingsvariabelen JAVA\_HOME en PATH zodat we de programma's van de JDK op gelijk welke plek op ons systeem te kunnen gebruiken.

- Download de installatiebestanden van JDK 11 van de website <http://java.oracle.com>. Ga naar de downloadpagina en kies de versie die overeenkomt met je platform<sup>1</sup>.

<sup>1</sup> Het versienummer kan verschillen van hetgeen in de afbeelding wordt weergegeven. Installeer gewoon de laatste versie van JDK 11 die op dit moment beschikbaar is.



Product / File Description	File Size	Download
Linux	147.4 MB	<a href="#">jdk-11.0.1_linux-x64_bin.deb</a>
Linux	154.09 MB	<a href="#">jdk-11.0.1_linux-x64_bin.rpm</a>
Linux	171.43 MB	<a href="#">jdk-11.0.1_linux-x64_bin.tar.gz</a>
macOS	166.2 MB	<a href="#">jdk-11.0.1_osx-x64_bin.dmg</a>
macOS	166.55 MB	<a href="#">jdk-11.0.1_osx-x64_bin.tar.gz</a>
Solaris SPARC	186.8 MB	<a href="#">jdk-11.0.1_solaris-sparcv9_bin.tar.gz</a>
Windows	150.98 MB	<a href="#">jdk-11.0.1_windows-x64_bin.exe</a>
Windows	170.99 MB	<a href="#">jdk-11.0.1_windows-x64_bin.zip</a>

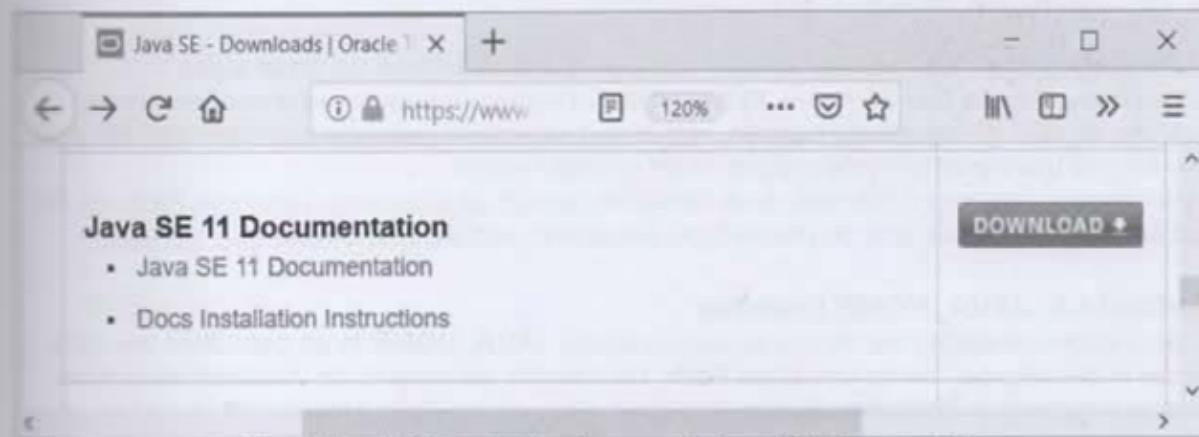
Afbeelding 10: Downloadpagina van de JDK

- Voer het installatieprogramma uit en gebruik hierbij telkens de standaardinstellingen.

### Opdracht 2: De JDK-documentatie installeren

In deze opdracht gaan we de documentatie bij de JDK lokaal installeren zodat we die steeds ter beschikking hebben, ook als we niet verbonden zijn met het internet.

- Haal de JDK-documentatie van de website <http://java.oracle.com>. Selecteer *Download bij Java SE 11 Documentation*.



Afbeelding 11: Downloadpagina van de Java-API-documentatie

- Pak het bestand **jdk-11.x.y\_doc-all.zip<sup>1</sup>** uit in een lokale map.
- Open het bestand ..\docs\index.html en maak eventueel een snelkoppeling naar dit bestand op het bureaublad of in het Start-menu.

<sup>1</sup> x en y staan voor het versienummer.



Afbeelding 12: Java-API-documentatie

## 2.3 De omgevingsvariabele JAVA\_HOME

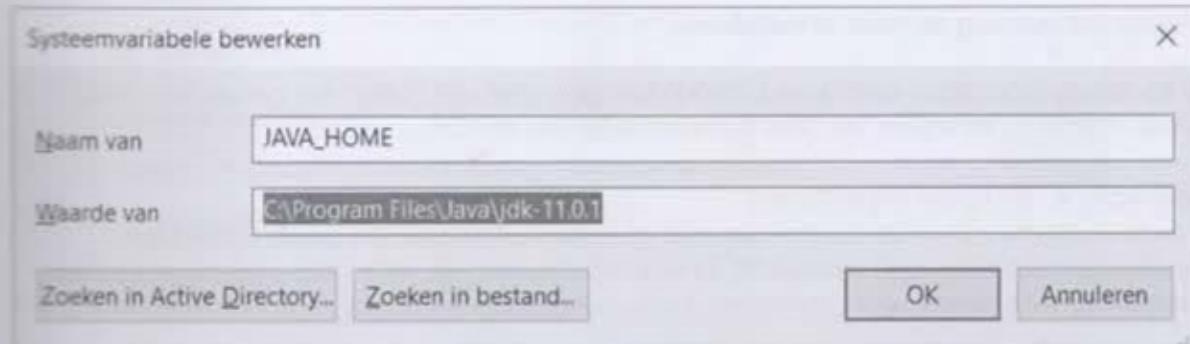
De geïnstalleerde JDK bevat allerlei programma's om onder andere code te compileren, documentatie te generen enzovoort. Deze programma's bevinden zich in de map **bin** van de installatiemap. Om deze programma's te kunnen gebruiken dienen we het besturingssysteem op de hoogte te brengen van de locatie van deze programma's. Dit gebeurt door het pad van de JDK-programma's toe te voegen aan de algemene omgevingsvariabele **PATH**. Indien we in een consolevenster de naam van een programma intikken zal het besturingssysteem namelijk dit programma zoeken in alle mogelijke paden die zijn opgenomen in deze omgevingsvariabele.

Dit toevoegen doen we evenwel in twee stappen. Eerst definiëren we onze eigen omgevingsvariabele met de naam **JAVA\_HOME**. Deze variabele verwijst naar de plaats waar we de JDK geïnstalleerd hebben. Het is een algemeen gekende variabele die ook door vele andere Java-gerelateerde programma's wordt gebruikt. In een tweede stap gebruiken we deze variabele binnen de algemene variabele **Path** om het pad aan te geven waar zich de uitvoerbare bestanden van de JDK bevinden.

### Opdracht 3: JAVA\_HOME instellen

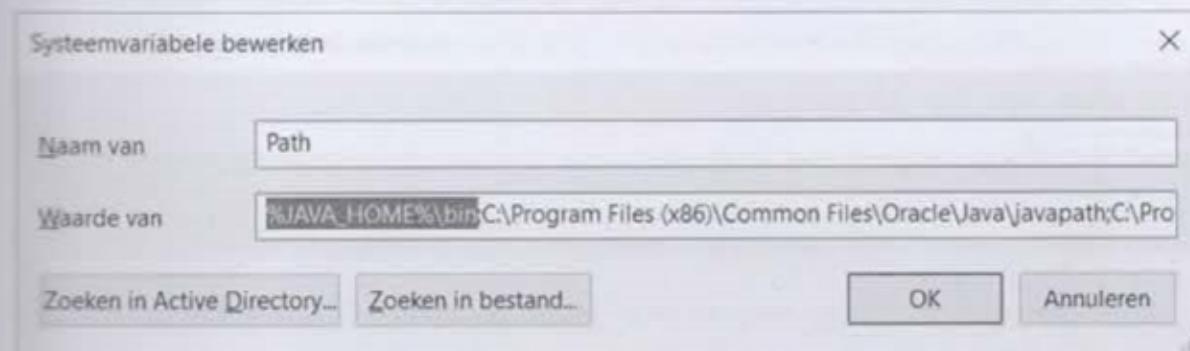
In deze opdracht stellen we de omgevingsvariabele **JAVA\_HOME** in en gebruiken we deze verder in de definitie van de variabele **Path**. De uitleg is gebaseerd op Windows-systemen, maar ook bij andere besturingssystemen kan je deze variabelen instellen.

- Zoek de locatie op waar je de JDK geïnstalleerd hebt. Bij Windows-systemen is dat doorgaans in de map **C:\Program Files\Java**. Het pad van de JDK kan bijvoorbeeld als volgt zijn: **C:\Program Files\Java\jdk-11.0.1**.
- Voeg bij de instellingen van het besturingssysteem de omgevingsvariabele **JAVA\_HOME** toe met het pad van de installatiemap van de JDK. Bij Windows-systemen open je hiervoor de **Windows-instellingen** en vervolgens zoek je naar "omgevingsvariabelen". Je voegt daar dan de volgende systeemvariabele toe:



Afbeelding 13: Systeemvariabele in Windows

- Wijzig vervolgens de systeemvariabele **Path** en plaats het pad `%JAVA_HOME%\bin` aan het begin van deze variabele. In Windows worden paden gescheiden door een puntkomma (;), bij op Unix gebaseerde systemen (zoals Linux, Mac OS) is dat een dubbele punt (:). Het %-teken duidt aan dat we gebruikmaken van een andere variabele in plaats van een letterlijke waarde.



Afbeelding 14: Systeemvariabele 'Path' bewerken

- Sluit alle configuratievensters en open een consolevenster. In Windows kan dat onder andere via het programma **cmd**.
- Voer in het consolevenster het volgende commando uit:

```
javac -version
```

The screenshot shows a 'Opdrachtprompt' window. The command 'javac -version' is entered and the output 'javac 11.0.1' is displayed. The window title is 'Opdrachtprompt'.

```
C:\Users\info>javac -version
javac 11.0.1
C:\Users\info>
```

- Indien dit programma werkt en het versienummer van de JDK geeft, zijn de omgevingsvariabelen correct ingesteld.

## 2.4 Ontwikkelomgevingen

De programmacode van Java kan geschreven worden in om het even welke tekstverwerker (zoals Notepad). Om efficiënter te werken, zijn er echter speciale ontwikkelomgevingen (*Integrated Development Environment* of IDE) te verkrijgen die een aantal taken kunnen automatiseren. Veel gebruikte IDE's zijn *Eclipse*, *IntelliJ IDEA* en *NetBeans*.

In de volgende opdrachten installeren we *Eclipse* en *IntelliJ IDEA*. Voor het verdere verloop

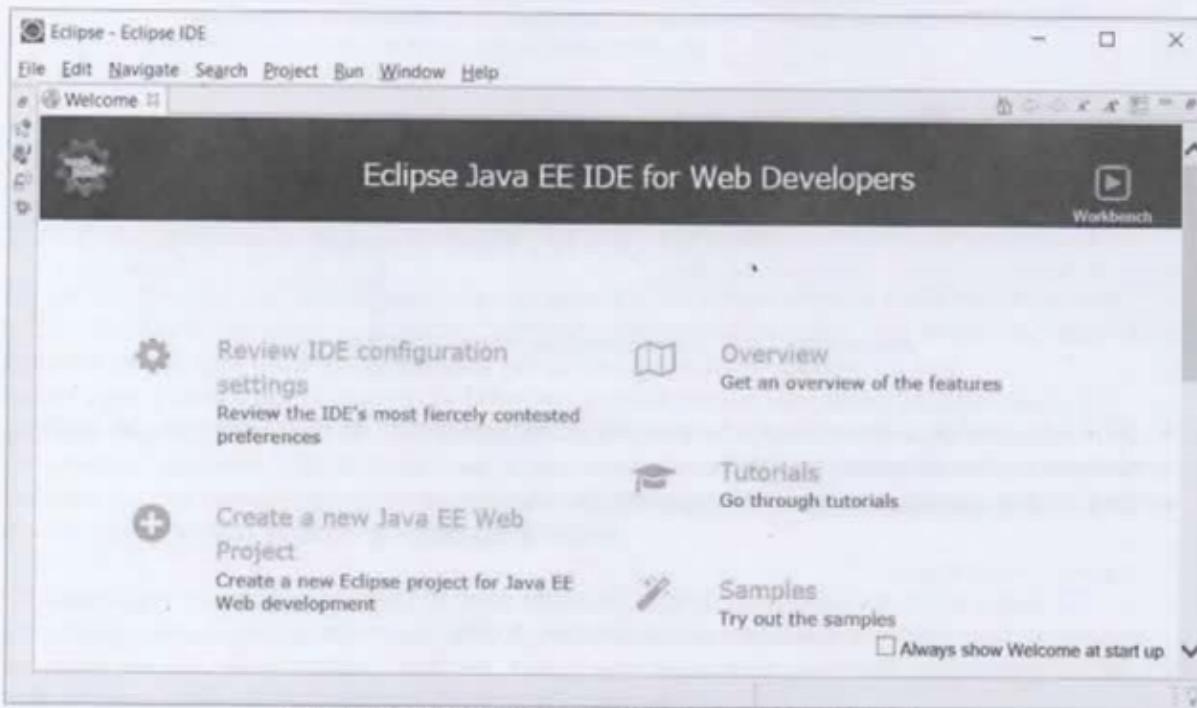
volstaat het een van de twee te installeren.

De gedetailleerde uitleg over deze ontwikkelomgevingen valt buiten het bestek van deze cursus. Hiervoor verwijzen we naar documentatie van de IDE.

#### Opdracht 4: Eclipse installeren

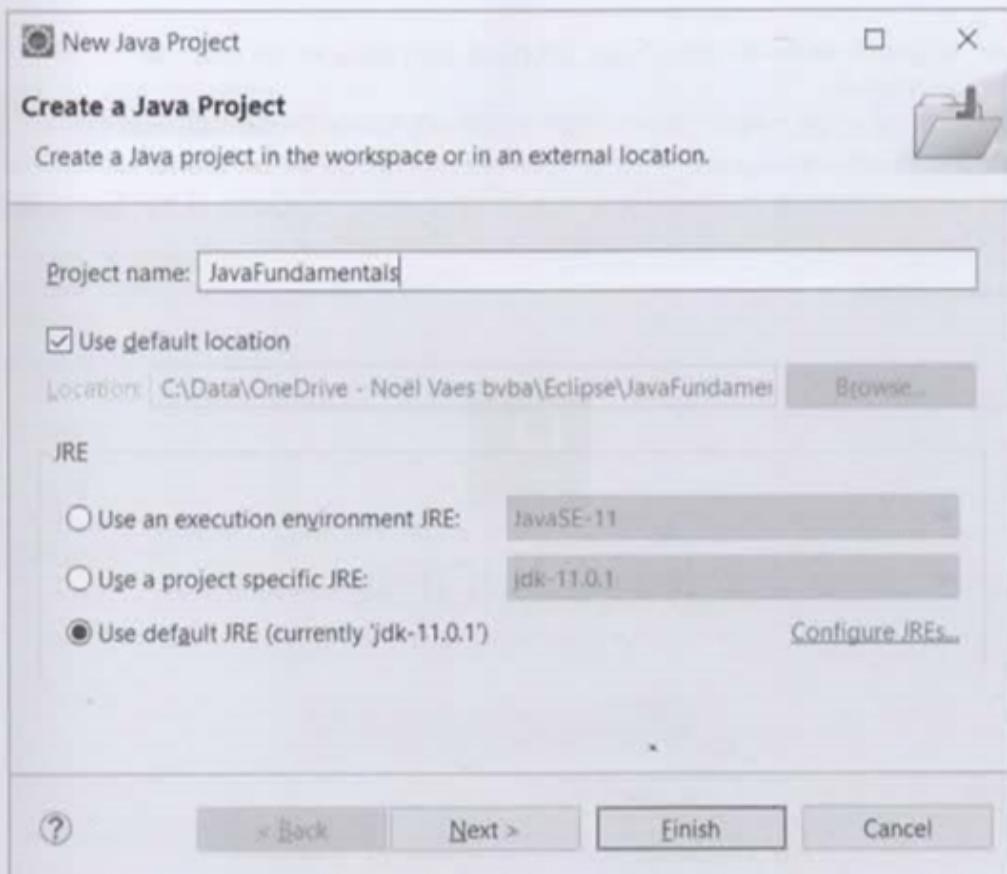
In deze opdracht zullen we *Eclipse* van het internet downloaden en lokaal installeren. Vervolgens maken we een eenvoudig Java-project waarin we later onze oefeningen zullen maken.

- Open volgende website: [www.eclipse.org](http://www.eclipse.org) en ga naar de downloadpagina.
- Selecteer eventueel je besturingssysteem en download **Eclipse IDE**. Kies de 64-bit versie.
- Download het bestand en start het installatieprogramma. Kies tijdens de installatie voor **Eclipse IDE for Java EE Developers**.
- Accepteer vervolgens de voorgestelde opties.
- Start *Eclipse* op via de snelkoppeling op je bureaublad of via het *Start*-menu.



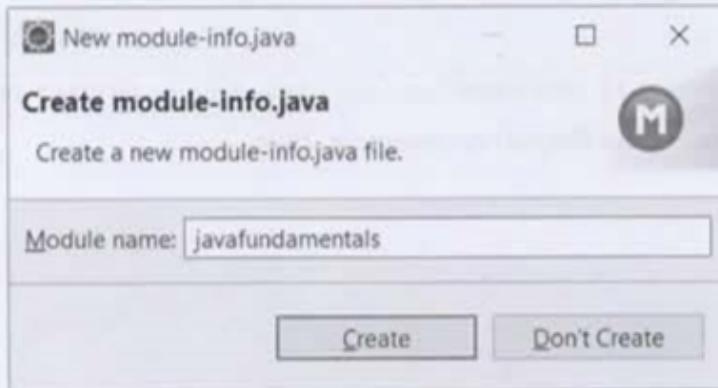
Afbeelding 15: Eerste scherm van Eclipse na installatie.

- Selecteer **File->New->Java Project**.
- Geef het project een naam (*JavaFundamentals*) en selecteer de juiste JRE (*Use default JRE*).



Afbeelding 16: Selectie van de JRE bij de aanmaak van een nieuw project

- Klik vervolgens op **Finish**.
- Geef de modulenaam op : **javafundamentals**



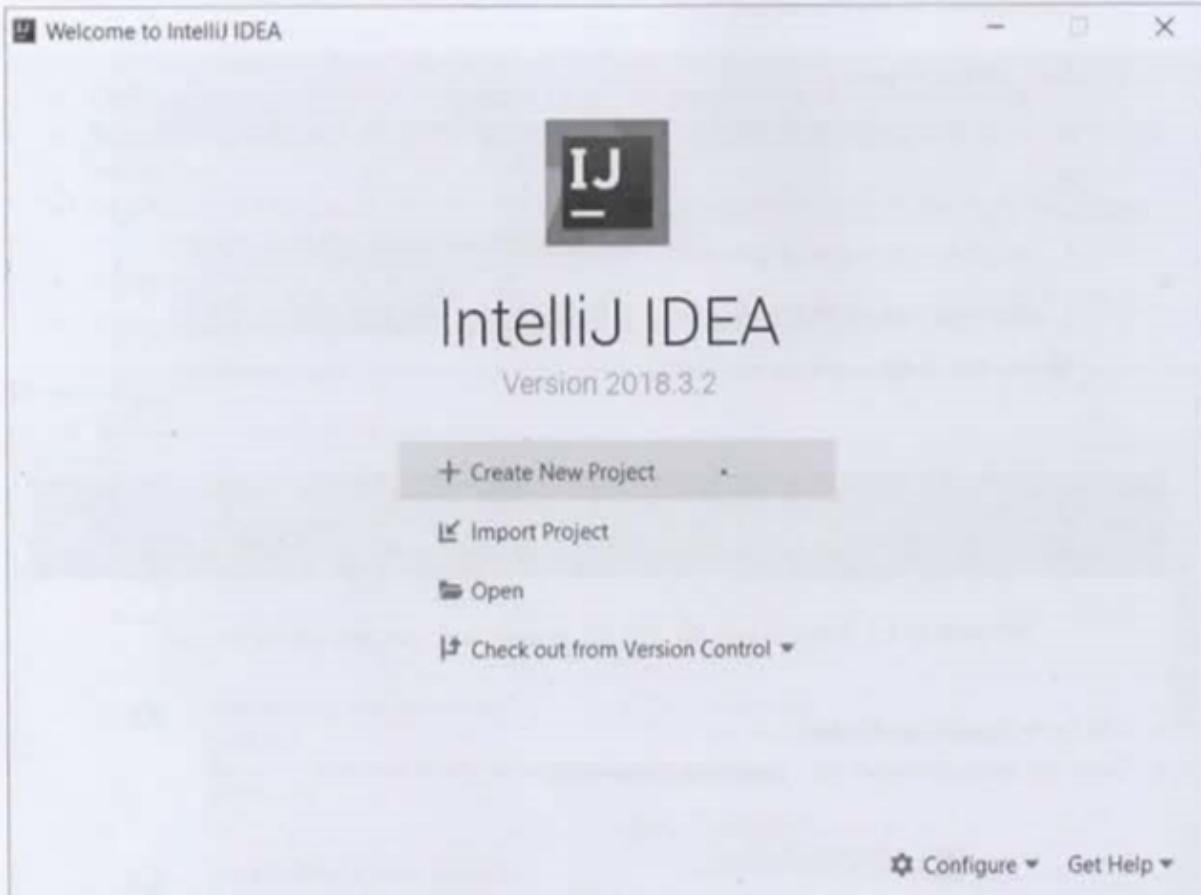
Afbeelding 17: De module-naam

- Klik op **Create**
- Bij de eventuele vraag naar de verandering van perspectief klik je gewoon op **OK**.
- Het project bevat een bestand **module-info.java** waarvan we later de juiste betekenis zullen zien.

#### Opdracht 5: IntelliJ IDEA installeren

In deze opdracht zullen we *IntelliJ IDEA* van het internet downloaden en lokaal installeren. Vervolgens maken we een eenvoudig Java-project waarin we later onze oefeningen zullen maken.

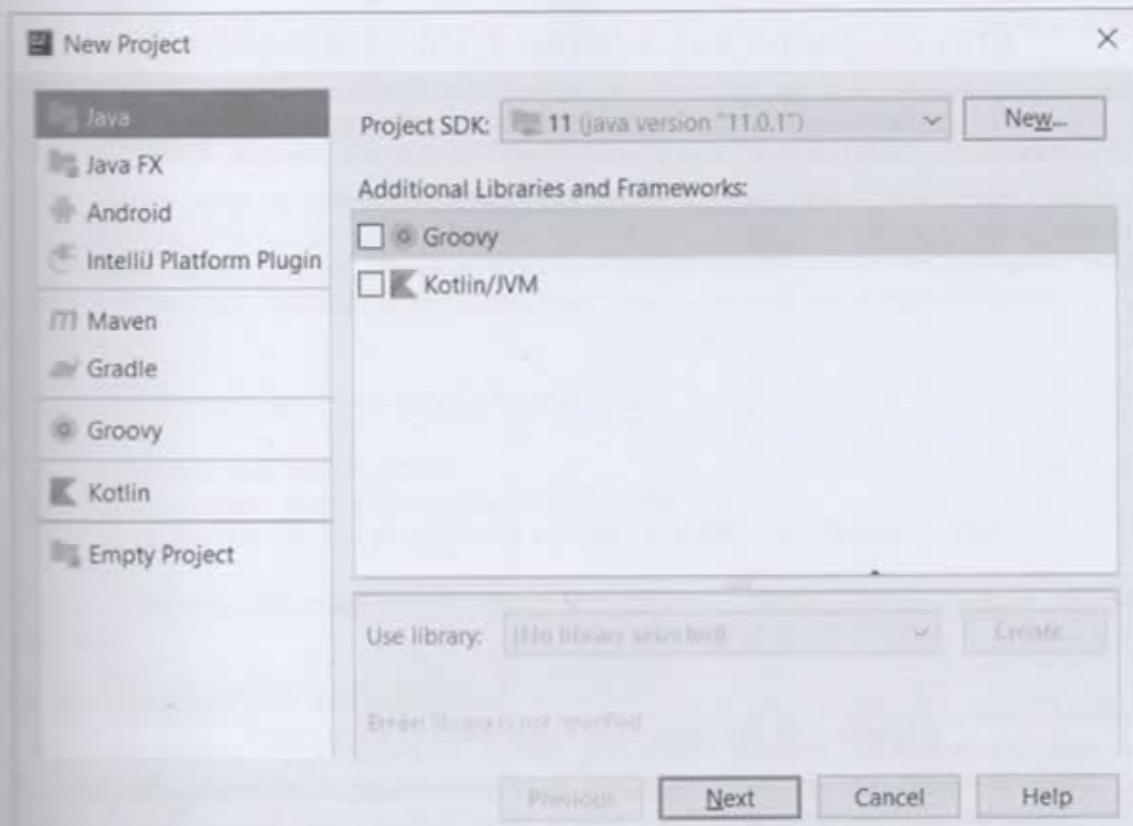
- Open volgende website: <http://www.jetbrains.com/idea/> en ga naar de downloadpagina.
- Selecteer het juiste besturingssysteem en download de *Community Edition*<sup>1</sup>.
- Voer het installatieprogramma uit en kies hierbij voor de **64-bit launcher**.
- Start het programma op via de snelkoppeling op het bureaublad of het Start-menu.



Afbeelding 18: Openingsscherm van IntelliJ IDEA na de installatie

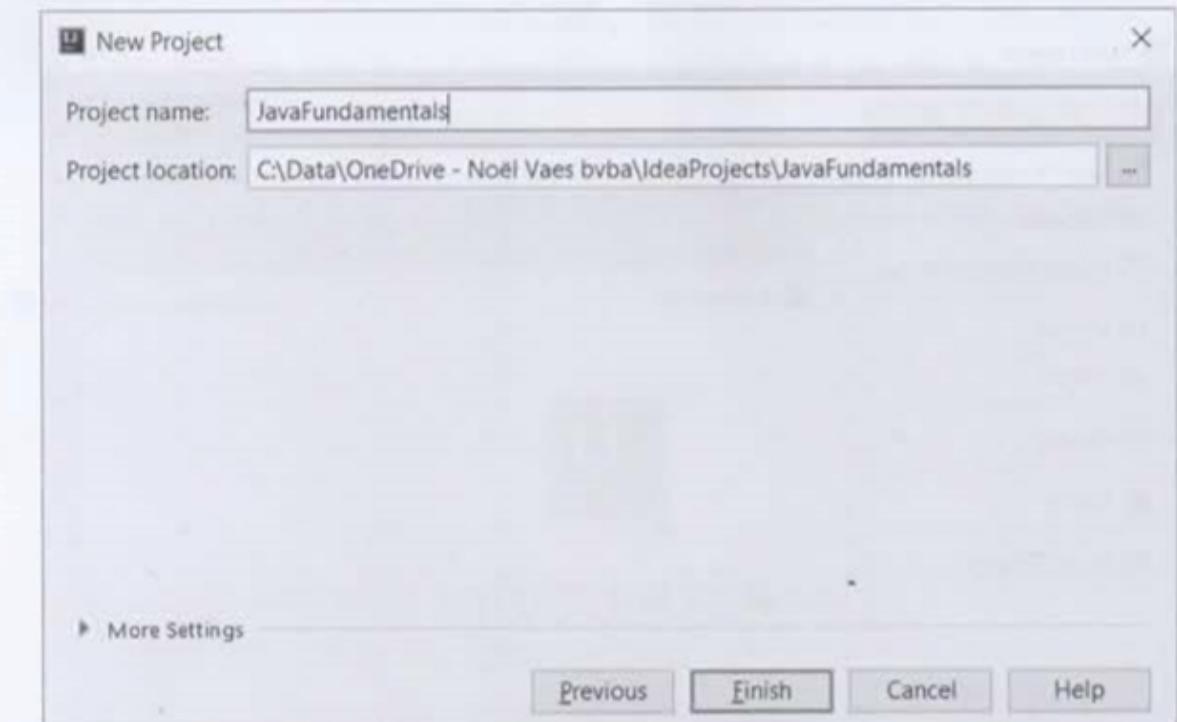
- Selecteer **Create New Project** en selecteer **Java**.

<sup>1</sup> Indien mogelijk de *Ultimate Edition* maar deze is beperkt in de tijd en vereist een licentie voor langer gebruik.



Afbeelding 19: Keuze van de JDK bij IntelliJ IDEA

- Bij **Project SDK** klik je op de knop **New** en vervolgens kies je **JDK**. In het volgende scherm geef je het pad op van de JDK die je op je systeem geïnstalleerd hebt. Op Windows is dat pad doorgaans **C:\Program Files\Java\jdk-11.x.y**.
- Na terugkeer naar het bovenstaande scherm klik je op **Next** en in het volgende scherm nogmaals.



Afbeelding 20: Het project een naam geven bij IntelliJ IDEA

- Geef het project een naam (bijvoorbeeld **JavaFundamentals**) en klik op **Finish**.

## 2.5 Samenvatting

In dit hoofdstuk hebben we de noodzakelijke toepassingen en documentatie geïnstalleerd voor de ontwikkeling van Java-programma's:

1. De *Java Development Kit (JDK)*.
2. De Java-API-documentatie voor lokaal gebruik.
3. Een *Integrated Development Environment (IDE)*.

Hiermee kunnen we nu verder aan de slag.

## Hoofdstuk 3: Mijn eerste Java-toepassing

### 3.1 Inleiding

In dit hoofdstuk gaan we ons eerste Java-programma schrijven. Het is zowat een traditie bij iedere opleiding in een programmeertaal om het eerste programma de tekst "*Hello World!*" op het scherm te laten tonen. Om deze traditie in ere te houden en om erbij te horen, zullen we onze eerste Java-toepassing de wereld laten begroeten met deze legendarische woorden.

Een Java-programma schrijven gebeurt in drie fasen:

1. Het maken van de **broncode**.
2. Het compileren van de broncode tot **bytecode**.
3. Het **uitvoeren** van het programma met de *Java Virtuele Machine (JVM)*.

We gaan deze drie stappen doorlopen en nadien gaan we dieper in op de opbouw van het programma.

### 3.2 De broncode schrijven

De broncode bestaat uit gewone tekst en kan geschreven worden met een eenvoudige tekstverwerker zoals *Kladblok*. Het is evenwel makkelijker gebruik te maken van een aangepaste IDE omdat deze op allerhande vlakken ondersteuning geeft bij het maken en onderhouden van de broncode. In het vorige hoofdstuk hebben we zo'n IDE geïnstalleerd.

Het broncodebestand wordt weggeschreven in gewoon tekstformaat en dient de extensie *.java* te hebben. Doorgaans wordt in een IDE dit broncodebestand weggeschreven in een submap met de naam **src** (*source*). Om de code uniek te maken wordt deze tevens geplaatst in een pakket waarvan de naam uniek is. De pakketnaam wordt weerspiegeld in een reeks submappen van de map **src**.

#### Opdracht 1: De broncode schrijven

In deze opdracht gaan we de broncode van ons eerste programma maken. We zullen dit broncodebestand ook in een afzonderlijk pakket steken.

- Maak in je IDE een nieuw pakket aan met de naam **hello**. Selecteer hiertoe de map **src** en kies vanuit het contextmenu **New->Package**. Geef als naam van het pakket "**hello**". Zie hoe er in de map **src** (de map die de broncode bevat) een nieuwe submap met de naam **hello** wordt toegevoegd.
- Maak in je IDE vervolgens een nieuwe Java-klasse aan en geef deze de naam **HelloWorldApp**. Selecteer hiertoe de map **src/hello** en kies vanuit het context-menu **New->Class (of Java Class)**. Het broncodebestand zal dan de naam **HelloWorldApp.java** krijgen.
- Tik de volgende Java-broncode in (let op hoofdletters en kleine letters!):

```
/* This Java application shows the text 'Hello World!' on the
screen. */
package hello;

public class HelloWorldApp {
    public static void main(String[] args) {
```

```

    System.out.println("Hello World!"); //Show the text.
}
}

```

### 3.3 De broncode compileren

In de tweede fase wordt de broncode gecompileerd tot *bytecode*. Zoals reeds eerder gezegd, is Java zowel een gecompileerde als geïnterpreteerde programmeertaal. In de eerste stap wordt de Java-broncode gecompileerd of vertaald naar een platformonafhankelijke *bytecode*. Deze *bytecode* zou men kunnen beschouwen als de binaire code of de machinetaal van een virtuele processor: de *Java Virtual Machine* of *JVM*. Tijdens de uitvoering van het programma wordt deze *bytecode* stap voor stap geïnterpreteerd en uitgevoerd.

De *bytecode* wordt weggeschreven in een bestand met extensie **.class**.

We kunnen het broncodebestand manueel compileren met de toepassing **javac** die deel uitmaakt van de JDK. Bij gebruik van een IDE zal deze het broncodebestand laten compileren door de compiler.

Bij *Eclipse* bevindt het *bytecode*-bestand zich in de map **bin**. Bij *IntelliJ IDEA* is dat de map **out/production/JavaFundamentals**. En ook hier zien we een submap die overeenkomt met de pakketnaam.

#### Opdracht 2: De broncode compileren

In deze opdracht gaan we de broncode van ons eerste programma compileren naar *bytecode*. Naargelang de IDE gebeurt dit enigszins op een andere wijze.

- *Eclipse*: het bestand wordt automatisch gecompileerd bij het wegschrijven. We hoeven hier dus niets extra te doen.
- *IntelliJ IDEA*: we dienen het project te compileren via het menu: **Build->Make Project** of **Build->Compile 'HelloWorldApp.java'**.
- Zoek via het bestandssysteem in de projectmap het bestand **HelloWorldApp.class**.

**Opmerking:** Indien de broncode fouten bevat, zal de compiler dit melden. Pas de broncode aan totdat de compilatie feilloos gebeurt.

Ons eerste Java-programma is nu af en is klaar om uitgetest te worden.

### 3.4 De bytecode uitvoeren

De gecompileerde *bytecode* kan maar uitgevoerd worden als ze geïnterpreteerd wordt door de Java-*interpreter*: **java**. Deze *interpreter* zelf is niet platformonafhankelijk. Ieder platform (hardware + besturingssysteem) heeft zijn eigen *interpreter* of *JVM* die aangepast is aan het platform.

De *interpreter* is te vinden in de submap **bin** van de installatiemap van de JRE of JDK.

We kunnen het programma nu uitvoeren door de *interpreter* op te roepen en hierbij de volledige klassennaam mee te geven.

```
java hello.HelloWorldApp
```

Bij gebruik van een IDE kan dit makkelijk gebeuren via het menu **Run**.

### Opdracht 3: Het programma uitvoeren

In deze opdracht gaan we de *bytecode* uitvoeren met de *Java-interpreter*. We zullen dit doen via de IDE.

- Selecteer in het menu **Run->Run**.



- Applaus voor onszelf!!!

## 3.5 De opbouw van het programma

Onze eerste toepassing *Hello World* bevat slechts enkele regels broncode. We zullen nu stap voor stap verklaren wat die regels te betekenen hebben.

```

1./* This Java application shows the text 'Hello World!' on the
   screen. */
2.package hello;
3.public class HelloWorldApp {
4.    public static void main(String[] args) {
5.        System.out.println("Hello World!"); //Show the text.
6.    }
7.}

```

### 3.5.1 Commentaar in Java-code

De eerste regel van ons programma begint met commentaar.

Het is een goede programmeertechniek om je broncode te doorspekken met heel wat commentaar. Zo kan je zelf achteraf beter achterhalen wat die code nu ook alweer betekent en bovendien maak je het je collega's heel wat makkelijker als zij aanpassingen moeten doen aan jouw code terwijl jij op de Canarische Eilanden ligt.

Java kent drie soorten commentaar:

1. **/\* comment \*/**  
Dit is de standaard commentaar zoals we die ook kennen in andere talen als C/C++. Alles tussen /\* en \*/ wordt genegeerd door de compiler. Deze methode wordt vooral gebruikt bij langere blokken commentaar, verspreid over meerdere tekstregels. Regel 1 bevat zo'n commentaarregel.
2. **/\*\* documentation \*/**  
Dit soort commentaar is bedoeld om achteraf automatisch documentatie te genereren met de JAVADOC-tool. De compiler zelf negeert alles wat tussen /\*\* en \*/ staat. In het hoofdstuk over JAVADOC gaan we hier verder op in.

### 3. // comment

In dit geval negeert de compiler alles wat achter // komt tot aan het einde van de regel. Deze methode wordt veel gebruikt bij korte stukjes commentaar achter een programmaregel zoals in regel 5 van ons programma.

Commentaar, en ook de code wordt doorgaans in het Engels geschreven. Dit is namelijk de standaardtaal voor programmeurs.

### 3.5.2 Het pakket definiëren

Met de regel `package hello;` geven we aan dat deze code tot een bepaald pakket behoort. De code die bij elkaar hoort, wordt vaak in eenzelfde pakket gestopt. Na compilatie komen de gegenereerde klassenbestanden terecht in submappen die de naam van het pakket weerspiegelen. Zo kwam het bestand `HelloWorldApp` terecht in de submap `hello`. Het is tevens gebruikelijk de broncode te plaatsen in een submap die overeenkomt met de pakketnaam.

```
..\JavaFundamentals\src\hello\HelloWorldApp.java
```

Pakketten worden ook gebruikt om klassen een unieke naam te geven. De volledige klassennaam is gelijk aan de pakketnaam plus de korte naam van de klasse. Om ambiguïteit omtrent pakketnamen te voorkomen, gebruikt men unieke internetdomeinnamen. Vermits de domeinnaam `noelvaes.eu` geregistreerd is, kan men deze naam als pakketnaam gebruiken:

```
package eu.noelvaes.examples;
```

Het broncodebestand en klassenbestand wordt daarbij geplaatst in de submappen die de pakketnaam weerspiegelen. Ieder punt in de pakketnaam komt hierbij overeen met een submap:

```
..\src\eu\noelvaes\examples\HelloWorldApp.java
```

### 3.5.3 De klasse definiëren

Java is een objectgeoriënteerde programmeertaal. In plaats van te werken met procedures (zoals C) werkt Java met objecten van een bepaalde klasse.

Een object is eigenlijk een verzameling van gegevens en methoden om met die gegevens om te gaan. Een klasse is een soort blauwdruk van een object.

Aan objecten en klassen wordt in het verder verloop van de cursus nog uitvoerig aandacht besteed.

De klasse wordt gedefinieerd in regel 3 met het woord `class` gevolgd door de naam van de klasse `HelloWorldApp`.

Wat er allemaal in die klasse zit, wordt gedefinieerd tussen de twee accolades.

```
public class HelloWorldApp {  
}
```

We hebben dus een nieuwe klasse gemaakt met de naam `HelloWorldApp`.

**Belangrijk:** Het bronbestand dat de code voor een klasse bevat, heeft bij voorkeur dezelfde naam als de klasse, met extensie `.java`. Bij publieke klassen is dit overigens verplicht. Het gebruik van de klassennaam maakt het onder andere makkelijker het broncodebestand van

een bepaalde klasse terug te vinden. Daarom dat we het bestand hebben weggeschreven met de naam **HelloWorldApp.java**. Na de compilatie wordt namelijk een **class**-bestand gemaakt dat dezelfde naam heeft als de klasse die in het bronbestand gedefinieerd wordt.

**Opgelet:** Java is hoofdlettergevoelig: **HelloWorldApp** is niet hetzelfde als **helloworldapp!**

### 3.5.4 De methode main()

De Java-interpreter start de uitvoering van het programma door de methode `main()` van de applicatieklasse aan te roepen. Iedere applicatie moet dus zo een methode `main()` hebben. Het is als het ware de ingangspoort (*entry point*) van de toepassing.

Bij de definitie van de klasse moeten we dus ook die methode `main()` definiëren. Dit gebeurt in regel 4

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

Eenwoordje uitleg:

- `public` betekent dat de methode `main()` voor publiek gebruik is. Met andere woorden de methode `main()` kan van buitenaf aangeroepen worden.
- `static` betekent dat de methode gemeenschappelijk is voor alle objecten van deze klasse. Alle variabelen en methoden die gedefinieerd zijn met het woord `static` zijn gemeenschappelijk voor elke instantie van die klasse.
- `void` wil zeggen 'leeg' en duidt erop dat de functie na uitvoering geen waarde teruggeeft aan degene die ze heeft opgeroepen.
- `String[] args` is de parameter die de functie al dan niet meekrijgt als hij wordt aangeroepen. Deze parameter is een *array* van *strings*. De inhoud van deze *strings* is gelijk aan de *command-line* parameters bij het opstarten van het programma. Bij het opstarten van een toepassing kan men namelijk extra gegevens meegeven die het programma kan gebruiken. Deze gegevens worden onmiddellijk achter de naam van de toepassing gezet, gescheiden door spaties.

Voorbeeld: `java hello.HelloWorldApp param1 param2`.

In dit geval zal in de eerste string het woord **param1** zitten en in de tweede string het woord **param2**.

- `{ }`  De programmatappen die worden uitgevoerd als de functie `main()` wordt aangeroepen, worden gedefinieerd tussen de accolades.

Geen angst als deze uitleg nog wat verwarring kan veroorzaken. In de loop van de cursus wordt hier nog uitvoerig op ingegaan.

### 3.5.5 Het eigenlijke werk

Tot nu heeft ons programma nog niets gedaan. Het op het scherm brengen van de tekst **Hello World!** gebeurt in regel 5:

```
System.out.println("Hello World!");
```

In deze regel wordt de methode `println()` aangeroepen van het object `out` dat hoort tot het object `System`. Objecten die horen tot andere objecten duidt men aan door de naam van het bezittende object te nemen, gevolgd door een punt en vervolgens de naam van het object.

De methode `println()` toont een tekenreeks (*string*) op het scherm.

### 3.6 Samenvatting

In dit hoofdstuk hebben we onze eerste programma geschreven, gecompileerd en uitgevoerd. Hiermee hebben we kennisgemaakt met het ontwikkelproces en ook met een eerste stukje code. De details van deze code zullen we in de volgende hoofdstukken verder toelichten.