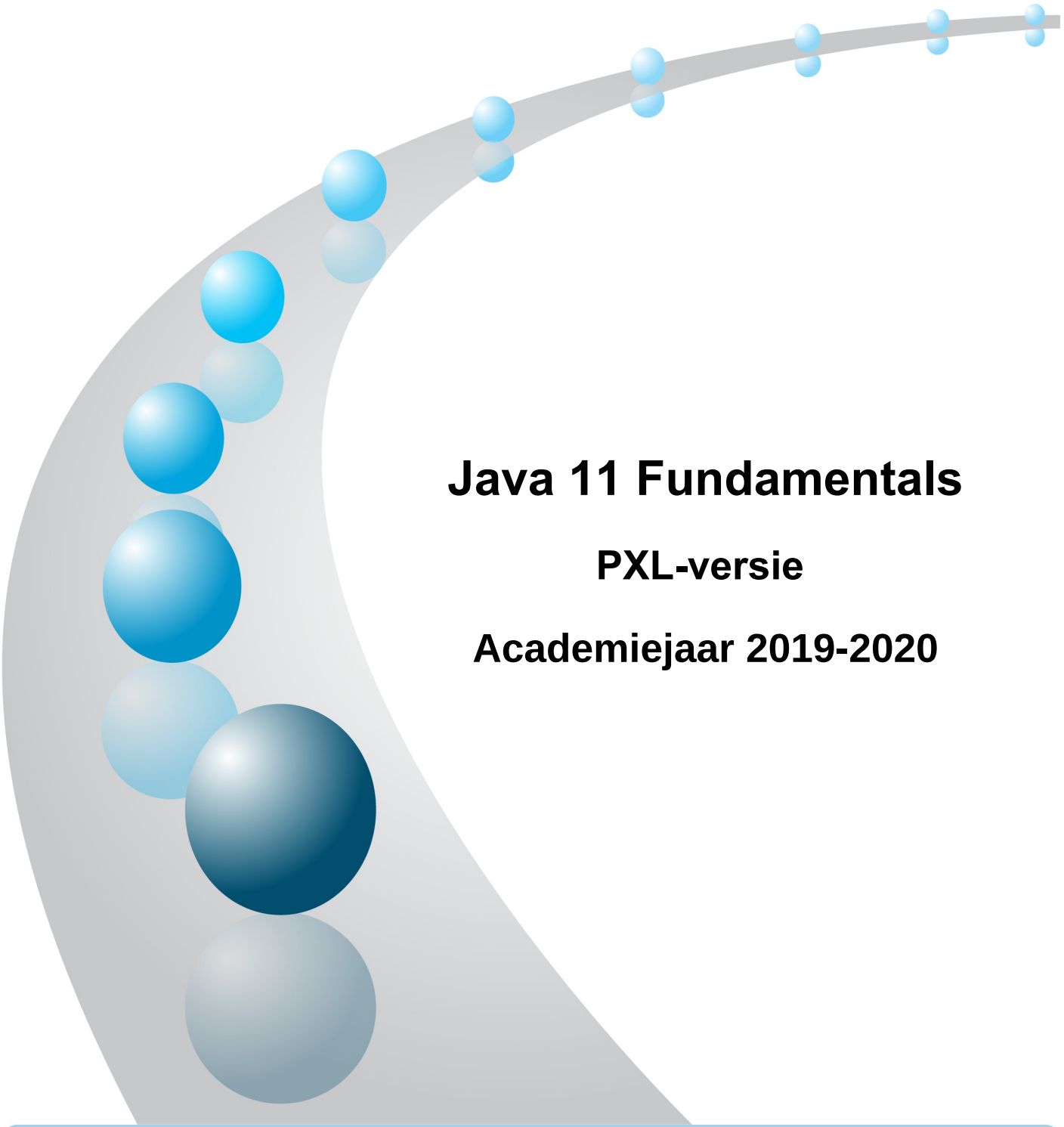
Noël Vaes

Java Trainer & Consultant



Java 11 Fundamentals

PXL-versie

Academiejaar 2019-2020

Roode Roosstraat 5
3500 Hasselt
België

+32 474 38 23 94
noel@noelvaes.eu
www.noelvaes.eu

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, opnamen 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
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.

30/11/2019

Copyright© 2019 Noël Vaes

Inhoudsopgave

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 niet meer weg te denken is 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.

Binairy code Intel
01100110
11010001
...

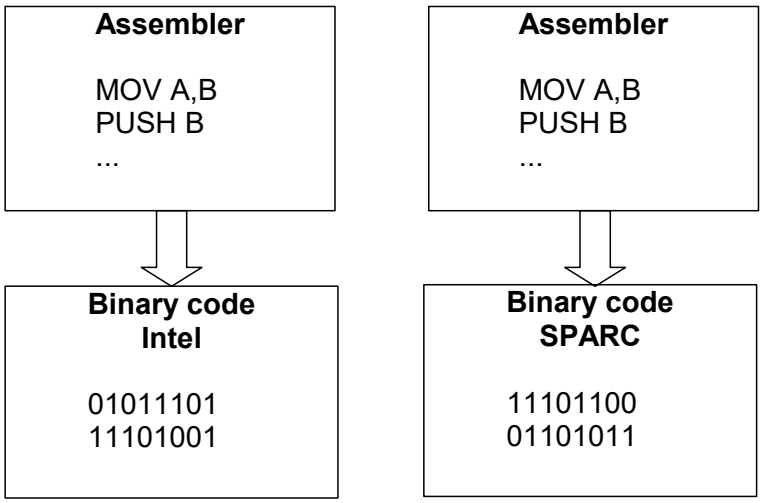
Binairy code SPARC
01011001
01010111
...

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*

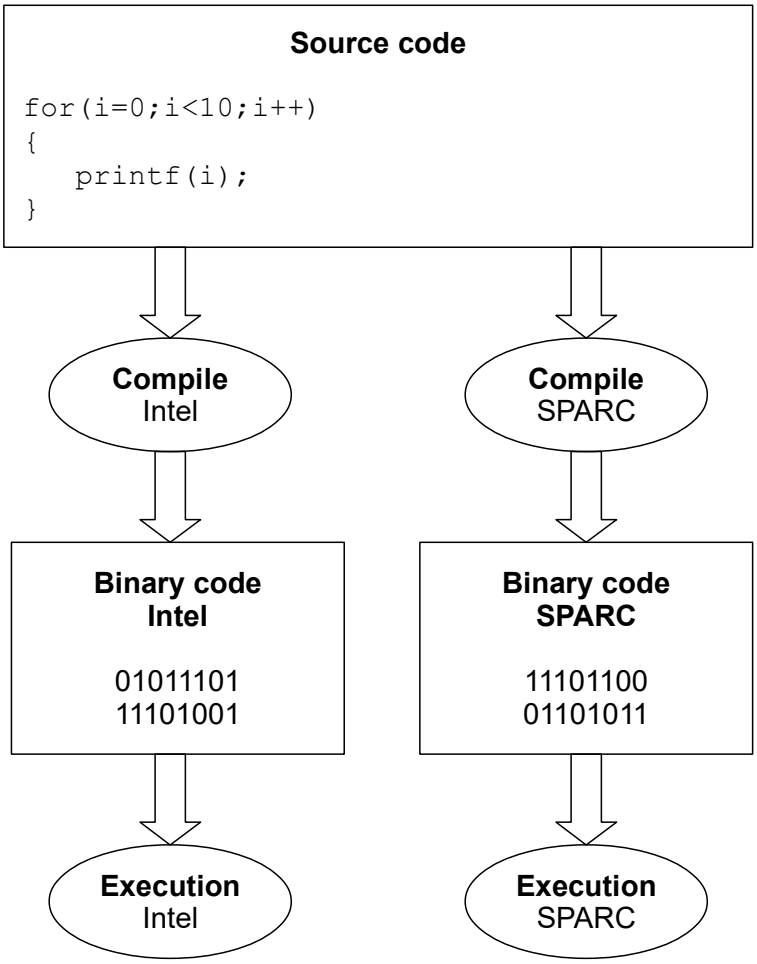
worden de binaire codes vervangen door gebruiksvriendelijkere woorden en symbolen. Het programma wordt geschreven in deze *Assembler*-codes en nadien vertaald in de overeenkomstige binaire codes.



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.

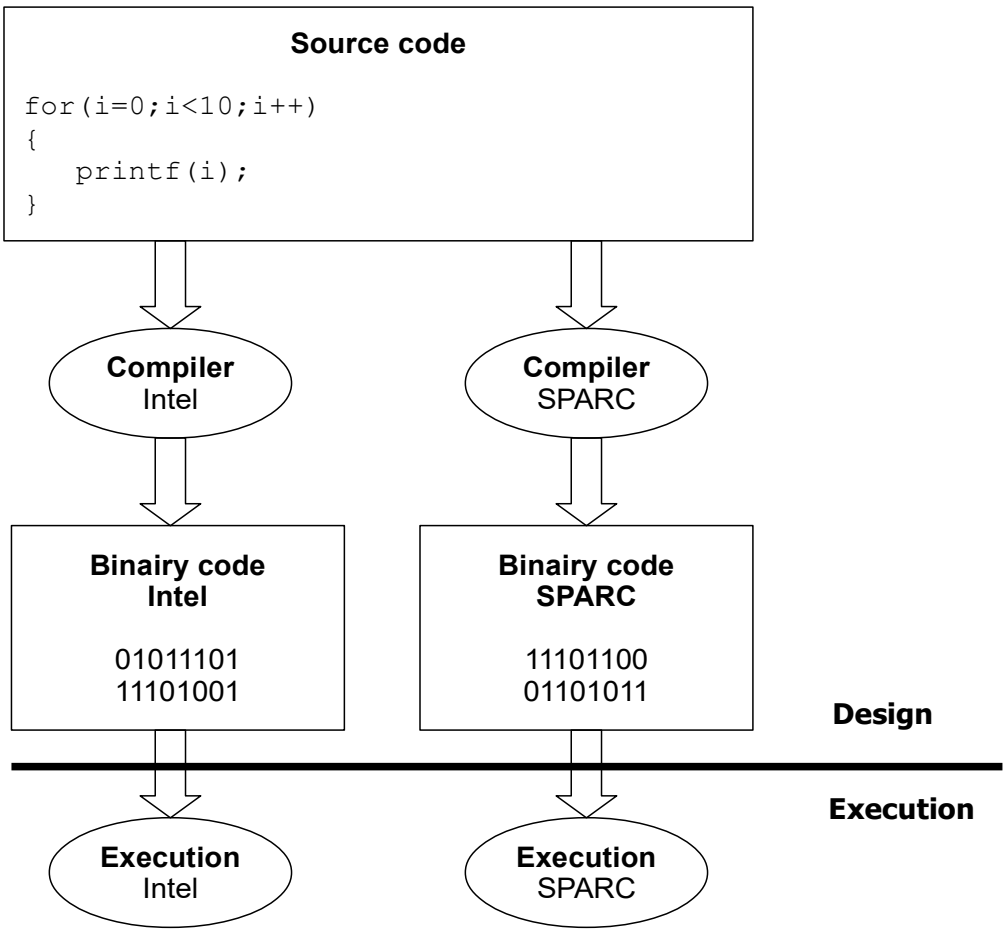
Op basis van dit vertaalmoment worden programmeertalen in twee groepen verdeeld:

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 'compileren' 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.

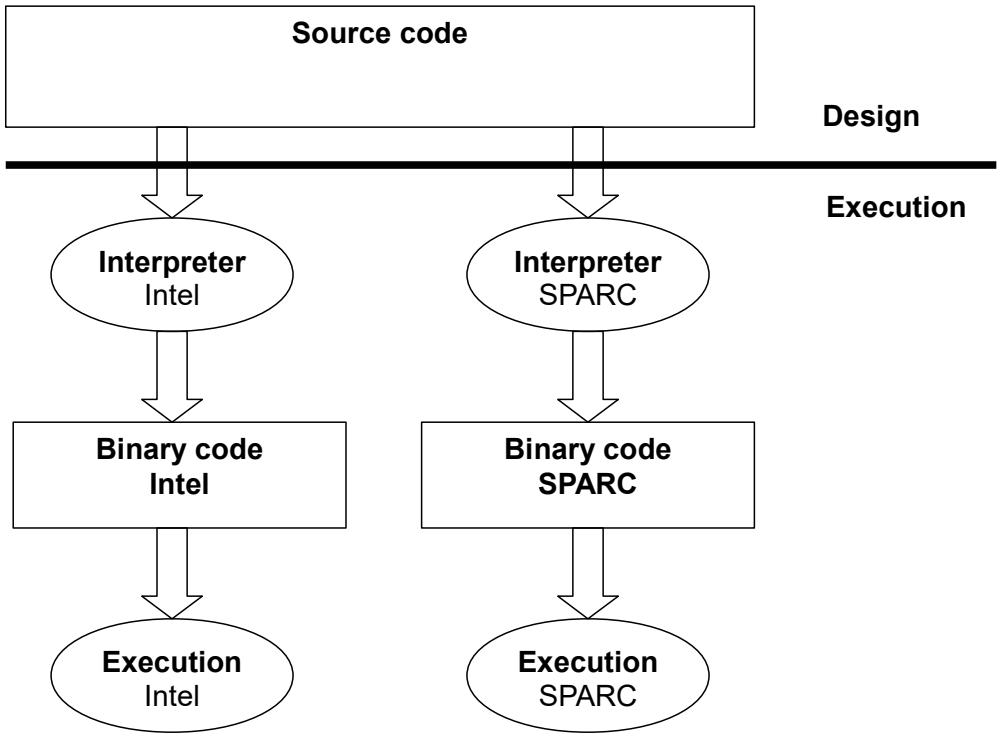
2. Voor elk besturingssysteem moet het programma afzonderlijk gecompileerd worden

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.

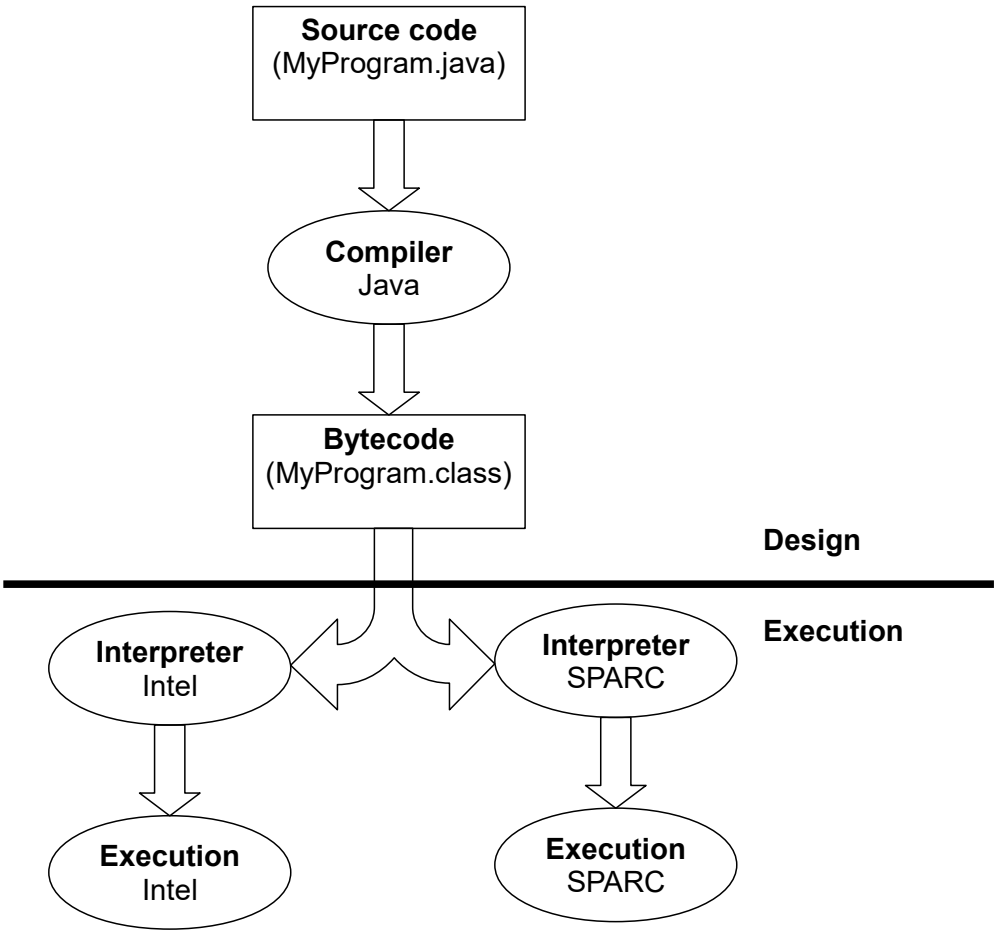
2. Het is moeilijk om de broncode te beschermen tegen illegaal gebruik. De programma's bestaan uit tekstbestanden die anderen naar believen kunnen kopiëren en aanpassen.

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 de compacte en efficiënte *bytecode* is Java **sneller** 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* (***J**ust **I**n **T**ime 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 zorgde 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 objectgeoriënteerd.

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.

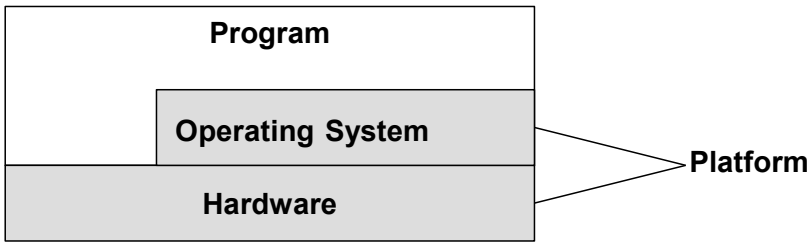
6. ***Multithreaded:*** Java biedt de mogelijkheid programma's te schrijven met meerdere uitvoeringsaders (*threads)*. Hierdoor kunnen in een Java-toepassing meerdere taken tegelijkertijd uitgevoerd worden.

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 ***Win****dows* en *In****tel***. Windows is het besturingssysteem dat gebruik maakt van de hardware op basis van Intel-processoren (of compatibele processoren).

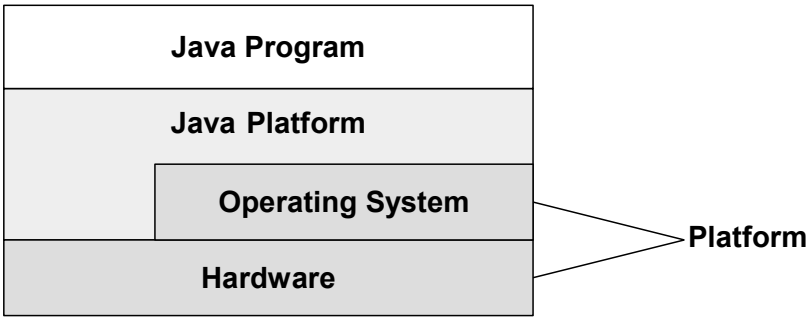


Afbeelding 7: Betekenis van een platform

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 wil 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.

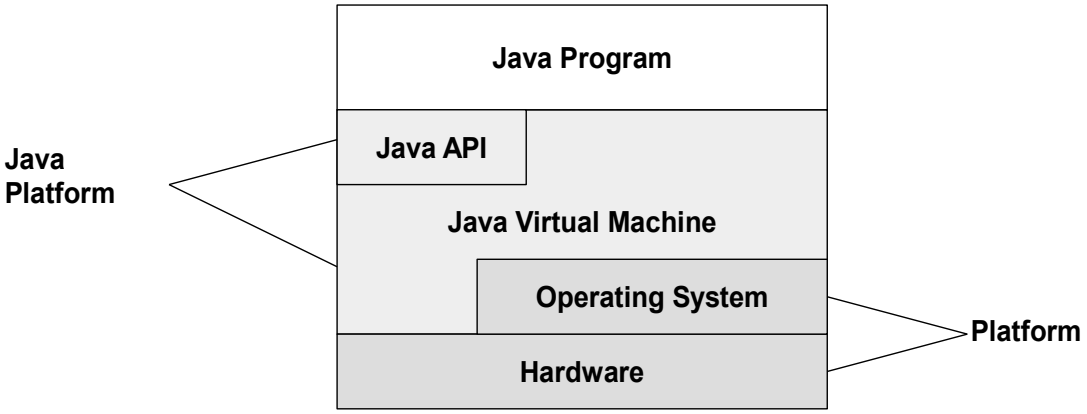


Afbeelding 8: Java als platform

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

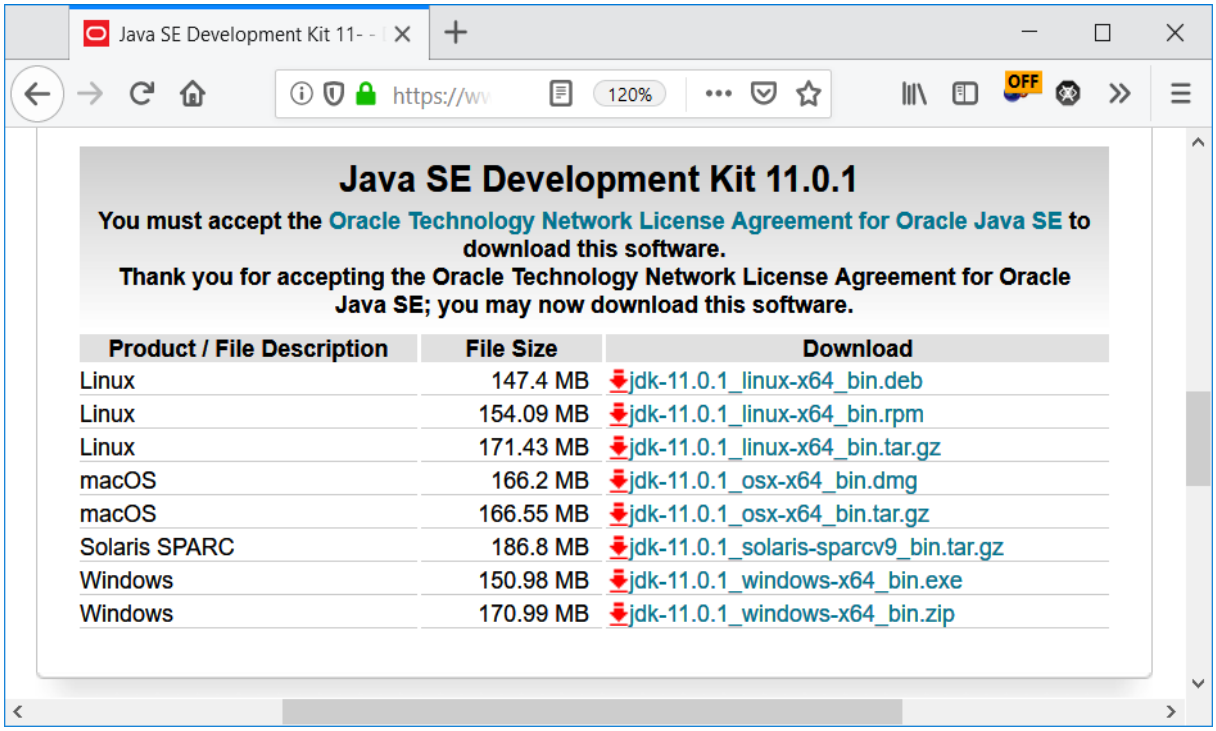
Aangezien de JVM zelf en de bijbehorende hulpprogramma's niet platformonafhankelijk zijn, dient men de juiste versie van de JDK te downloaden.

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 instellen zodat we de programma's van de JDK op om het even welke plek op ons systeem 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[1].

1 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.



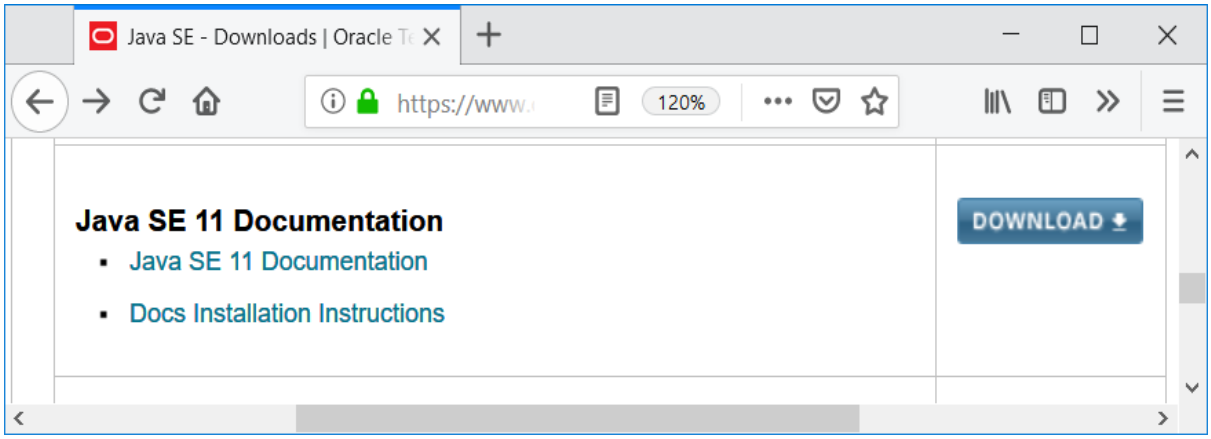
Afbeelding 10: Downloadpagina van de JDK

- 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***[1] 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.

1 x en y staan voor het versienummer.