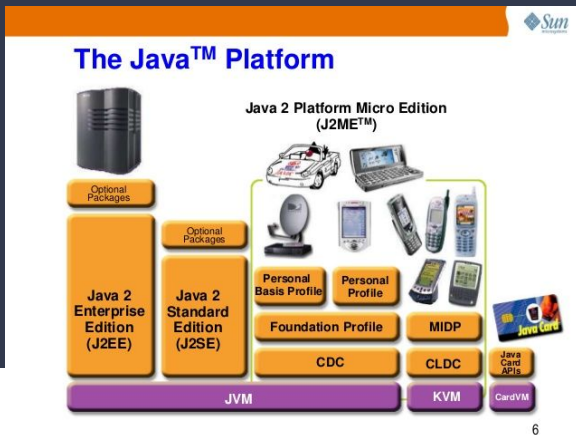


Java

De programmeertaal

A dark blue diagonal gradient bar that starts from the bottom left and extends towards the top right, covering the lower half of the slide.

De geschiedenis van Java



ORACLE®

De programmeertaal Java werd in 1995 ontwikkeld door het bedrijf **SUN**.

Java en zijn voorganger (OAK) waren aanvankelijk bedoeld om een **robuuste programmeertaal** te zijn voor consumenten logica.

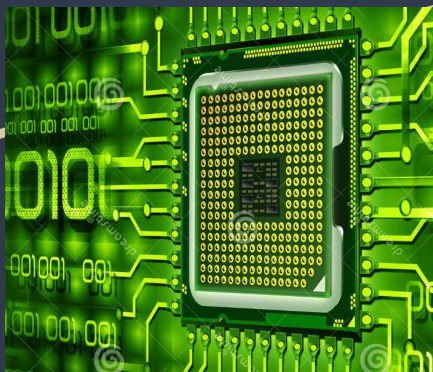
Java is **uitermate geschikt** voor een **groot netwerk** dat bestaat uit verschillende soorten computersystemen.

Dit vooral door zijn **platform onafhankelijke** karakter.

Java wordt momenteel vooral gebruikt voor het maken van enterprise-applicaties.

Java is zowel een **programmeertaal** als een **platform**.

Java als programmeertaal.



Java kan slechts **werken met binaire codes**.

Iedere **instructie** die Java uitvoert, is eigenlijk een **binair getal** dat opgeslagen is in het **werkgeheugen**.

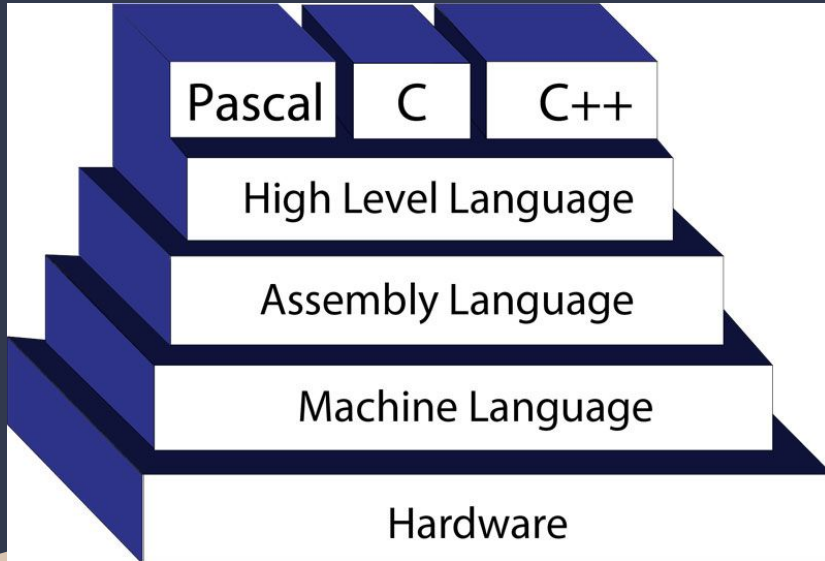
De **processor** haalt dit getal (instructie) uit het geheugen en **voert deze uit**.

Deze binaire codes en de overeenkomstige instructies zijn **specifiek voor** iedere **processor** of **processorfamilie**.

Zo heeft een processor van Intel een andere instructie dan de SPARC van SUN.

Beide zijn op binair niveau helemaal niet compatibel.

Soorten programmeertalen.



De **allereerste** programmeurs schreven **programma's** rechtstreeks in binaire code, ook wel **machinetaal** genoemd.

Dit soort programmeerwerk was **vrij omslachtig** en vooral ook **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. **Tweede generatie** programmeertaal genoemd.

Assembler

```
# fuzzification
1      slw      $6,$table_1,$4      # fuzzify inputs in the
2      slw      $7,$table_2,$5      # registers 4 and 5
# rule evaluation
3      rulev    $14,$6,$7,0,0      # rule R11
4      rulev    $8,$6,$7,0,1      # rule R21
5      rulev    $9,$6,$7,1,0      # rule R22
6      max      $13,$8,$9          # combine rules for output set LARGE
7      rulev    $8,$6,$7,0,2      # rule R31
8      rulev    $9,$6,$7,1,1      # rule R32
9      max      $12,$8,$9          # combine rules for output set HALF
10     rulev    $8,$6,$7,1,2      # rule R41
11     rulev    $9,$6,$7,2,0      # rule R42
12     max      $11,$8,$9          # combine rules for output set SMALL
13     rulev    $8,$6,$7,2,2      # rule R51
14     rulev    $9,$6,$7,2,1      # rule R52
15     max      $10,$8,$9          # combine rules for output set TINY
# loading constants
16     lui      $4,4097            # singletons of
17     lw       $15,1024($4)       # output fuzzy sets
...
# first set
22     blez     $10,$L8           # defuzzification performed
23     move     $3,$10            # using up to 5 multiplications
24     mult     $2,$10,$15        # to multiply singletons
# second set
25     $L8:     blez     $11,$L9   # and up to 10 additions:
26     addu     $3,$3,$11         # 5 to accumulate set values
27     mult     $4,$11,$16        # and up to 5 to accumulate
28     addu     $2,$2,$4          # products
# third set
...
# division
40     div      $2,$2,$15         # the output is determined by
```

```
L0: MOV    R1, #a      ; Address of a
    MOV    R2, #b      ; in R1, of b in R2

L1: LD     R3, (R1)     ; Inport bits in R3
    CMP    R3, #0      ; IF-condition
    BNE    L3          ;

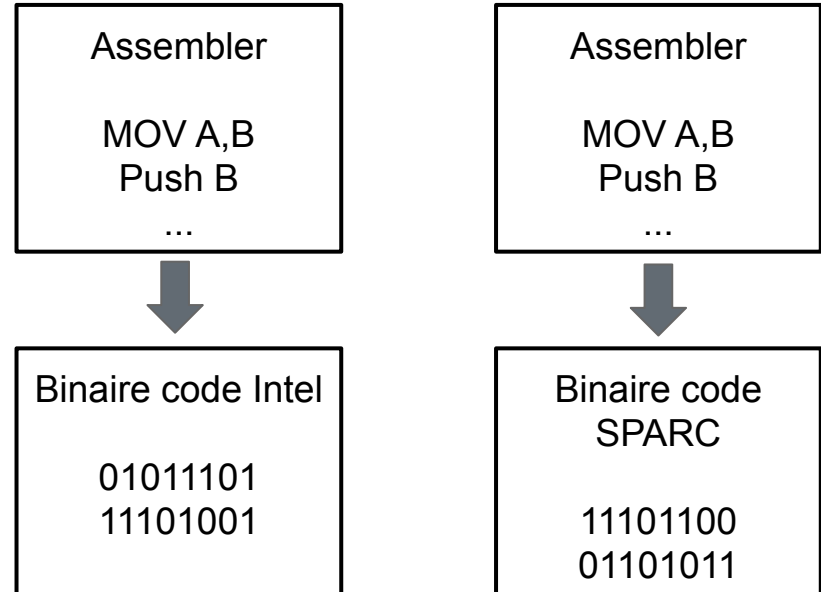
L2: MOV    R4, #1      ; IF-branch
    JMP    L4          ;

L3: MOV    R4, #0      ; ELSE-branch

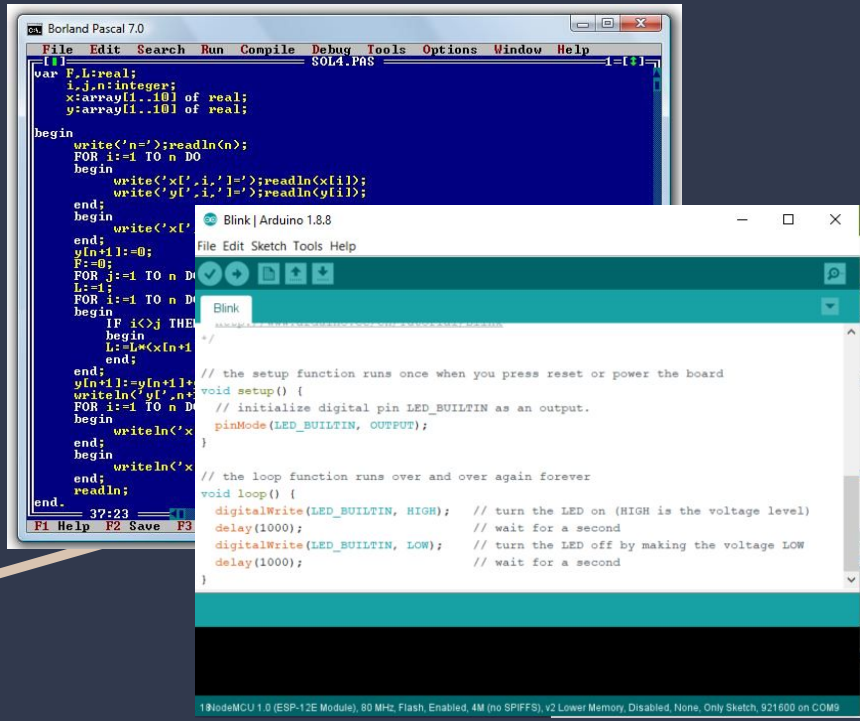
L4: ST     (R2), R4    ;
    JMP    L1          ;
```

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**.



Assembler vs 3de generatie programmeertalen.



- De Assembler-code voor verschillende processoren lijkt al meer op elkaar, toch is **Assembler** niet meer dan een **gebruiksvriendelijke voorstelling** van de binaire code.

→ Is dus geen echte programmeertaal.

- Bij **hogere programmeertalen** (C/C++, Visual Basic, Pascal, Cobol, ...) worden woorden in plaats van binaire codes gebruikt

→ Dit noemen we de broncode.

Hogere programmeertalen.

Sommige hogere programmeertalen (zoals C++/Java) zijn **overdraagbaar**.

Dit 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**.

Op basis van dat **vertaal moment** worden programmeertalen in **twee groepen** verdeeld:

1. **Gecompileerde** programmeertalen.
2. **Geïnterpreteerde** programmeertalen.

	A COMPILER	AN INTERPRETER
Input	... takes an entire program as its input.	... takes a single line of code, or instruction, as its input.
Output	... generates intermediate object code.	... does not generate any intermediate object code.
Speed	... executes faster.	... executes slower.
Memory	... requires more memory in order to create object code.	... requires less memory (doesn't create object code).
Workload	... doesn't need to compile every single time, just once.	... has to convert high-level languages to low-level programs at execution.
Errors	... displays errors once the entire program is checked.	... displays errors when each instruction is run.

Gecompileerde programmeertalen.

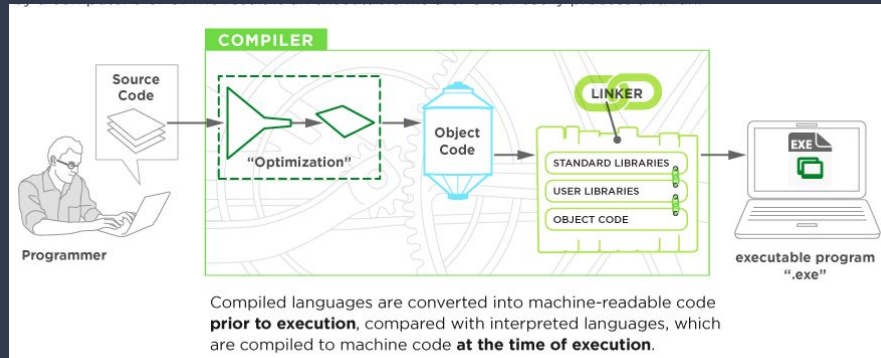
Bij gecompileerde programmeertalen wordt de **broncode** **weggeschreven** in een **tekstbestand**.

Deze broncode wordt **vervolgens vertaald** naar de **binaire objectcode** die wordt **weggeschreven** in een **uitvoerbaar binair bestand**.

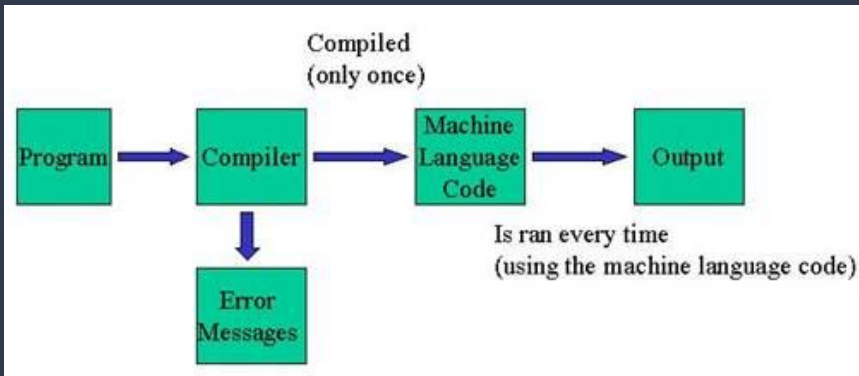
- Dit **proces** wordt ook wel **compileren** genoemd.
- Wordt gedaan door een **compiler**.

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

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



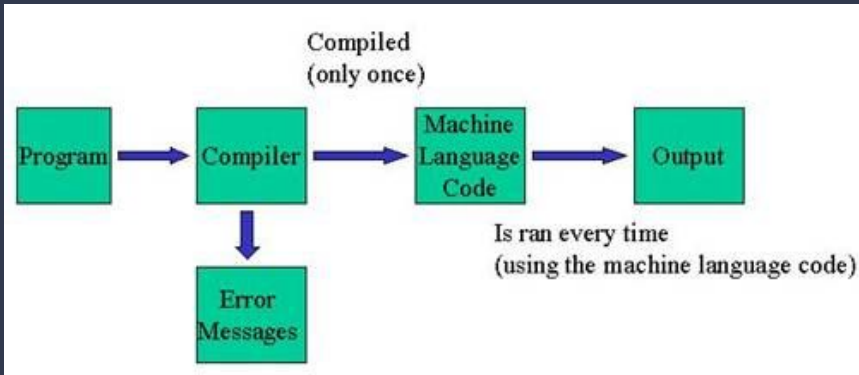
Voordelen gecompileerde programmeertalen.



Voordelen:

1. De **broncode** van gecompileerde talen is **overdraagbaar**.
2. Gecompileerde programma's **zijn snel** omdat de binaire code rechtstreeks kan worden uitgevoerd.
3. De **objectcode is binair** en kan dus moeilijk aangepast of gebruikt worden door anderen. De **broncode** is dus **goed beschermd**!

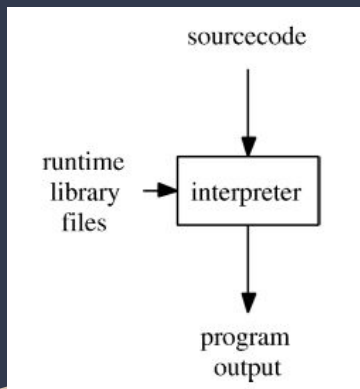
Nadelen gecompileerde programmeertalen.



Nadelen:

1. Voor elk type processor moet een afzonderlijk binair bestand (objectcode) gemaakt worden. De **uitvoerbare programma's** zijn **niet overdraagbaar**.
2. Voor **elk besturingssysteem** moet het programma **afzonderlijk gecompileerd** worden omdat de interactie met het besturingssysteem telkens anders is.
 - Zowel broncode als de objectcode zijn **afhankelijk van het besturingssysteem**.
1. De programma's moeten eerst gecompileerd worden vooraleer ze getest kunnen worden. **Na iedere aanpassing volgt nogmaals een compilatie**. Het testen en debuggen worden hierdoor **omslachtig** en **tijdrovend**.

Geïnterpreteerde programmeertalen.



Bij geïnterpreteerde programmeertalen wordt de **vertaling** gedaan **tijdens** de **uitvoering** van het **programma**.

De **sourcecode** wordt ook hier opgeslagen in een tekstbestand en tijdens de uitvoering van het programma worden de programmaregels **stap voor stap geïnterpreteerd en uitgevoerd**.

→ Dus **geen intermediair bestand** met objectcode!

Voordelen geïnterpreteerde programmeertalen.



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** (vertaling gebeurt door interpreter[**vb browser**]).

Nadelen geïnterpreteerde programmeertalen.

Nadelen:


1. Programma's **werken traag**, omdat alle programma **stappen 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**

```
> var a = {
  value: 2,
  toString:function(){
    return ++this.value
  }
}

if(a==3 && a==4){
  console.log('js is amazing')
}

js is amazing
```

Java versus andere programmeertalen.

Language Rank	Types	Spectrum Ranking
1. Python	  	100.0
2. C++	  	99.7
3. Java	  	97.5
4. C	  	96.7
5. C#	  	89.4
6. PHP		84.9
7. R		82.9
8. JavaScript	 	82.6
9. Go	 	76.4
10. Assembly		74.1

Java is iets of wat een buitenbeentje tussen de programmeertalen. Het is **zowel** een **gecompileerde** als **geïnterpreteerde** programmeertaal.

→ Het verenigt dus de voordelen van beide soorten.

Java wordt geschreven in een gewoon tekstbestand (**broncode**) met **extensie java** (vb **MyProgram.java**)

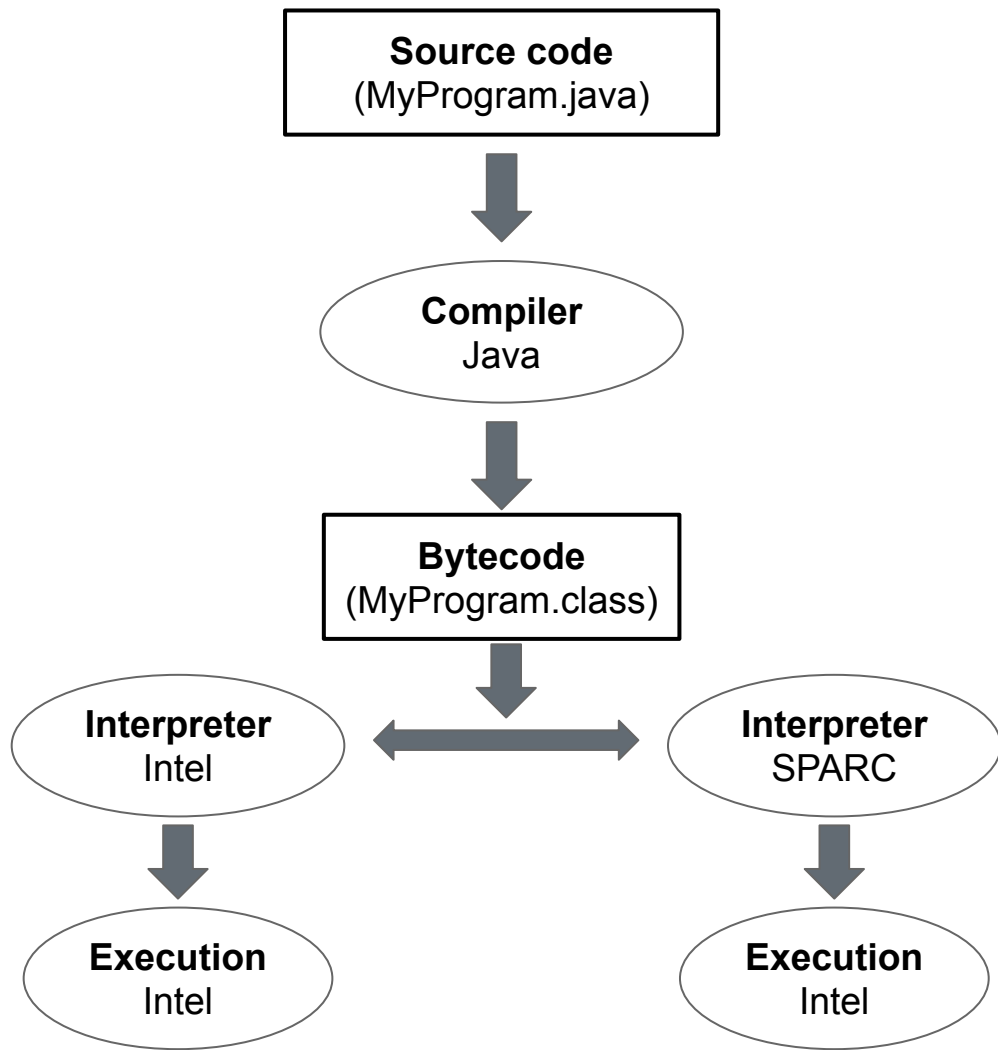
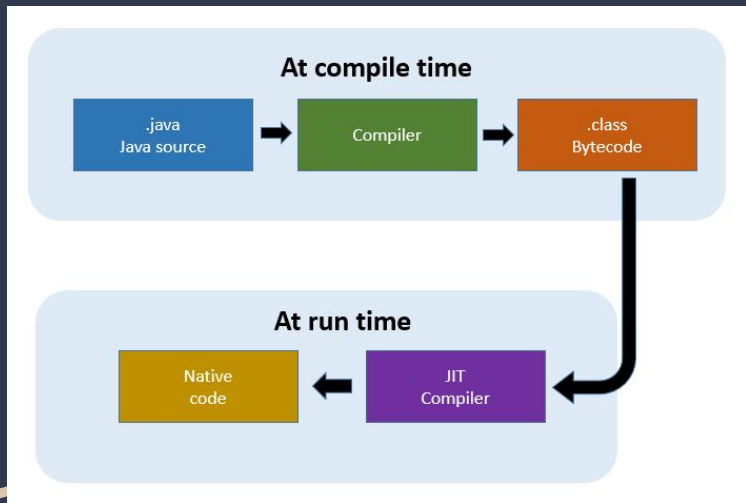
Deze **broncode** wordt **gecompileerd** naar de **binaire code** van een **virtuele machine** met een virtuele processor en besturingssysteem.

Dit **noemt** men de **bytecode**.

→ Dit word **opgeslagen** in een **bestand** met de extensie **class** [vb **MyProgram.class**]

→ Wordt **nadien geïnterpreteerd** en **uitgevoerd** door de **JVM**.

Hoe kan ik me dit voorstellen?



Voordelen Java.

1. Gecompileerde **Java-programma's** zijn **overdraagbaar**. De **bytecode is universeel** en kan door elke JVM gebruikt worden.
2. Vanwege **compacte** en **efficiënte bytecode** is Java **sneller** dan de meeste geïnterpreteerde talen.
3. De bytecode is **beter beschermd tegen illegaal gebruik** en aanpassingen.
4. Java is niet enkel **processor onafhankelijk** maar ook **platformonafhankelijk**.

Nadelen van Java?

Java is **trager dan pure gecompileerde programmeertalen** omdat de bytecode toch geïnterpreteerd moet worden.

Dit trachtte 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.

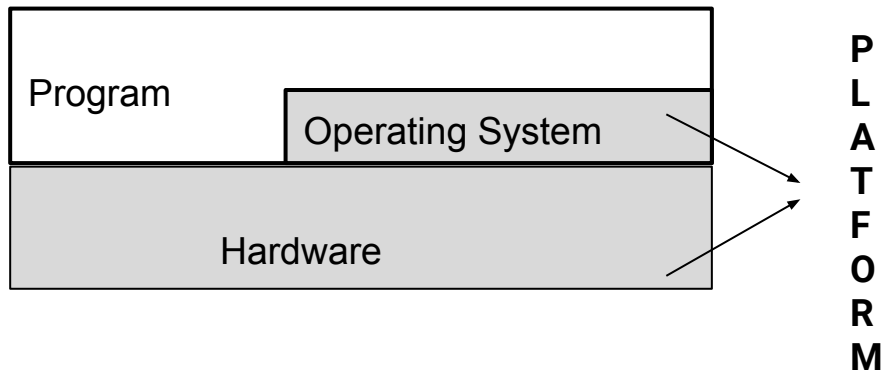
- Zorgt aanvankelijk voor de nodige vertraging.
- Laatste versies van Java zijn echter gebaseerd op de **HotSpot**-technologie.

PS: andere instructeurs dwingen mij om deze slide leeg te laten (Er zijn geen **nadelen**).

Wat is nu een platform?

Onder platform verstaan we de **combinatie** van **hardware** en een **besturingssysteem**.

→ Bekendste platform is het **WINTEL-platform** (Wintel is **samenvoeging** van **windows** en **intel**).



Gecompileerde programma's worden doorgaans gecompileerd voor een specifiek platform.

Java als platform

Het **Java-platform** is louter softwarematig en is gebouwd **bovenop** het **gewone platform**.

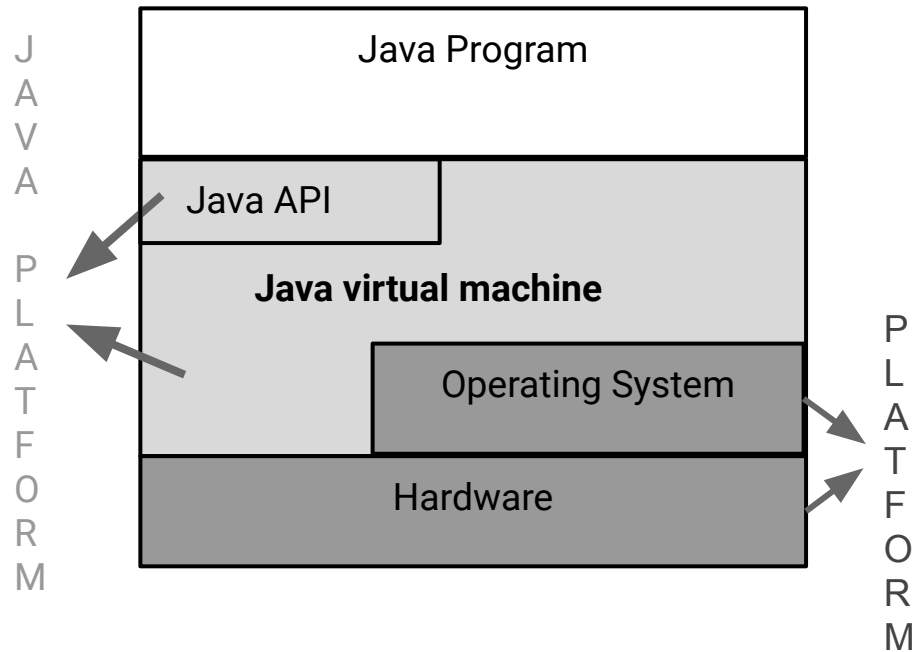
Wil zeggen dat het Java-platform abstractie maakt van het concrete hardwareplatform en de programmacode isoleert.

→ Net dit maakt Java platform onafhankelijk en overdraagbaar.

Dit impliceert wel dat het **Java-platform zelf niet platform onafhankelijk** is!

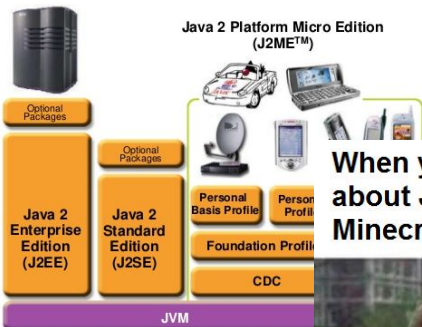
→ **Ieder platform** moet over zijn **eigen JVM** beschikken. Enkel de **Java programma's** zijn **platformonafhankelijk**.

Java platform schematisch

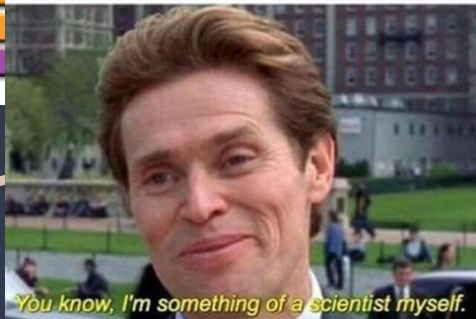


Soorten Java-toepassingen.

The Java™ Platform



When your teacher is talking
about Java and you remember
Minecraft was made with Java



- **Java-desktop applicaties:**

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.

- **Java-applets:**

Applet is het verkleinwoord van application. Een **applet** is dus een **kleine applicatie**. Deze applets worden **uitgevoerd binnen** de context van een **internetbrowser**.

- **Java-server applicaties:**

Dit zijn Java-applicaties die uitgevoerd worden op een (web)server.

<https://blog.newrelic.com/technology/what-you-can-do-with-java/>