Java Programming

# Inleiding algoritmes

## Algemene begrippen

|  |  |
| --- | --- |
| **3GL** | ***3****rd* ***G****eneration* ***L****anguage* |
| **bytecode** | *code, 2nd Generation Language* |
| **JVM** | ***J****ava* ***V****irtual* ***M****achine (Java software emuleren)* |
| **JDK** | ***J****ava* ***D****evelopment* ***K****it (om Java te programmeren, debuggen, ... bevat JRE)* |
| **JRE** | ***J****ava* ***R****untime* ***E****nvironment (Java programma’suitvoeren JVM + Java API)* |
| **API** | ***A****pplication* ***P****rogramming* ***I****nterface* |
| **opleidingshoofd** **IT** | *Koen Schram* |

**Machine Code** (1st Generation Language)

100101011011011101100001011101110110000101101110110000101101110110 000101101110110000101101110110000101101110110000101101110110000101 1011101100001001…

**Assembler** (2nd Generation Language)

MOV AX, 47104   
MOV DS, AX   
POP [3998], 36   
INT 32

**3GL** (3rd Generation Language) – Java, C, C++, Visual Basic, ...

Scanner scanner = **new** Scanner(System.**in**);   
**boolean** mooiWeer = scanner.nextBoolean();   
**if** (mooiWeer) {   
 System.out.println("**Laat je paraplu maar thuis!**");   
} **else** {   
 System.out.println("**Vergeet je paraplu niet!**");   
}

**Compileren**: omzetten van broncode naar binaire code waarbij de ***compiler*** eerst het hele programma scant en vertaalt en als een geheel naar machinecode omzet. vb. C

* Snelle uitvoering
* Broncode overdraagbaar
* Code beschermd
* Platform afhankelijk
* Debugging en testen vraagt extra stap

**Interpreteren**: omzetten van broncode naar binaire code waarbij de ***interpreter*** statement per statement vertaalt naar machinecode. vb. JavaScript

* Eenvoudig code aanpassen
* Onmiddellijk overdraagbaar (platform onafhankelijk)
* Tragere uitvoering
* Broncode onbeschermd

binaire code

broncode

## Java programma

Ontwikkeld door **James Gosling** in 1994, voormalig tewerkgesteld bij Sun wat nu overgenomen is door Oracle.

* Onmiddellijk overdraagbaar (platform onafhankelijk)
* Sneller dan geïnterpreteerd
* Broncode (min of meer) beschermd
* JVM nodig om de bytecode uit te voeren
* Trager dan gecompileerd

### Kenmerken

|  |
| --- |
| Geïnterpreteerd |
| Objectgeoriënteerd |
| Gedistribueerd |
| Platform onafhankelijk |
| Robuust |
| Multithreaded |
| Veilig |
| Snel |

*Java compiler*

**bytecode**

*MyProgram.class*

**broncode**

*MyProgram.java*

*Intel interpreter*

**UITVOEREN**

Een **algoritme** is een reeks instructies die uitgevoerd moeten worden om een bepaald doel te bereiken.

**Declareren** is het vermelden van variabelen en bepalen welk soort data ze gaan bevatten.

**int** som;

**Initialiseren** is een waarde geven aan een variabele.

**int** som = 56;

# Variabelen en operatoren

## Algemene begrippen

|  |  |
| --- | --- |
| **Variabele** | *Stukje geheugen waar je gegevens in kan stoppen (heeft een naam en bereik/scope)* |
| **Initialiseren** | *Een waarde toekennen aan een variabele* |
| **Declareren** | *Voordat je een variabele kan gebruiken deze een naam en type toekennen (\*)* |
| **literals** | *letterlijke waarden in de code* |

##### Regels voor naamgeving variabelen (\*)

|  |
| --- |
| enkel letters, cijfers, \_ en $ |
| naam mag niet met een cijfer beginnen |
| naam mag geen gereserveerd keyword zijn |
| naam moet uniek zijn binnen de scope van de variabele (unieke naam = identifier) |

## Gereserveerde sleutelwoorden

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **abstract** | **boolean** | **break** | **byte** | **case** |
| **catch** | **char** | **class** | **const** | **continue** |
| **default** | **do** | **double** | **else** | **extends** |
| **final** | **finally** | **float** | **for** | **goto** |
| **if** | **implements** | **import** | **instanceof** | **int** |
| **interface** | **long** | **native** | **new** | **package** |
| **private** | **protected** | **public** | **return** | **short** |
| **static** | **strictfp** | **super** | **switch** | **synchronized** |
| **this** | **throw** | **throws** | **transient** | **try** |
| **void** | **volatile** | **while** | **assert** | **enum** |

 Je kan meerde variabelen van hetzelfde type op één regel declareren.

int getalEen, getalTwee;

Het declareren en initialiseren van een **final** variabele (constante), deze wijzigt niet na initialisatie.

**final** int MAXIMUM\_TEMPERATUUR = 100;

**Casten** is specifiek een type converteren naar een ander type (bv. double naar int), dit kan mogelijks een verlies van precisie betekenen (in dit geval decimalen).

**double** getalEen = 1.7;  
**int** getalEenAlsInt = (**int**) getalEen;

System.out.println(getalEenAlsInt); //1

Een **char** kan op twee manieren een waarde toegekend worden, met een effectief karakter of met een overeenstemmende code uit de ASCII-tabel.

**char** letter = ‘A’;  
**char** letterAsci = 65;

## Operatoren

|  |  |
| --- | --- |
| **Modulo %** | *de rest van een deling* |
| **promoveren** | *+op promoveert een byte, short of char naar int* |
| **Relationele binaire operatoren** | *=> <= != ...* |
| **Logische binaire operatoren** | *|| &&* |
| **Logische unaire operator** | *=! (inverse)* |
| **Toekenningsoperatoren binair** | *= -= += \*= %= /= ...* |
| **Conditionele ternaire operatoren** | *= ? : vb.: (BMI >25) ? “te hoog” : “OK”* |

## Increment en decrement

|  |  |  |
| --- | --- | --- |
|  | pre | post |
| increment | **int** getal;  **int** i = 10;  getal = ++i;  //inhoud getal: 11  //inhoud i: 11 | **int** getal;  **int** i = 10;  getal = i++;  //inhoud getal: 10  //inhoud i: 11 |
| decrement | **int** getal;  **int** i = 10;  getal = --i;  //inhoud getal: 9  //inhoud i: 9 | **int** getal;  **int** i = 10;  getal = i--;  //inhoud getal: 10  //inhoud i: 9 |

# Uitdrukkingen, statements, codeblokken & constructies

Een **uitdrukking** is een geldige reeks van variabelen, operatoren en functieaanroepen dewelke resultaten in een enkelvoudige waarde. Een uitdrukking noemt men ook een **expressie**.

a = **15 \* b + c**;

if (**a < 120**){...

Een overzicht van statements, uitdrukkingen, ... enkelvoudige statements worden altijd afgesloten met een ;

**double** a; //declaratiestatement  
a = 15 \* Math.PI; //toekenningsstatement met een uitdrukking (15 \* Math.PI)

//if statement in de onderstaande codeblok **if** (a < 120) {  
 System.out.println(a); //methode aanroep  
}

De **scope** of **bereik** van een variabele is het gebied in de code/het programma waar de variabele gekend is. Een codeblok start met { en eindigt met }. In geneste codeblokken zijn de variabelen ook gekend.

## Constructies (programmaverloop statements)

### Conditie

Om enkel bepaalde stukken code uit te voeren als voldaan wordt aan een bepaalde conditie maakt men gebruikt van testen: if, if ... else, switch

**if** (condition) {  
 statements;  
} **else** {  
 statements;  
}

**if** (condition) {  
 statements;  
} **else if** (condition) {  
 statements;  
} **else if** (condition) {  
 statements;  
} **else** {  
 statements;  
}

Bij een **switch** kan break ook weggelaten worden, in dit geval spreekt men van een ***fall through*** en loopt de code over in de volgende case.

**switch** (waarde){  
 **case** literal1:  
 statements;  
 **break**;  
   
 **case** literal2;  
 statements;  
 **break**;  
  
 ...  
  
 **default**:  
 statements;  
}

### Iteratie

Om een codeblok te herhalen zolang een bepaalde uitdrukking waar is maakt men gebruik van lussen.

**while** (condition) { //de conditie wordt gecheckt  
 statements; //indien true, erna terug naar conditie check  
} //indien false, meteen stop

 Je kan een lus voortijdig afbreken door in de codeblok break te zetten, echter best enkel in een switch statement te gebruiken.

**do** {  
 statements; //de statements worden uitgevoerd  
} **while** (condition); //conditiecheck, indien waar terug naar statements

##### Verloop binnen een for loop

|  |
| --- |
| initialisation |
| condition check |
| statements |
| increment |
| condition check |

**for** (initialisation; condition; increment){  
 statements;  
}

### Sequentie

Opeenvolging van statements.

# Objectgeoriënteerd programmeren