*Programming Basics Samenvatting*

# Theorie

Java is ontstaan omdat deze betrouwbaar, objectgeoriënteerd en onafhankelijk van soort CPU was. Java is zowel een programmeertaal als een platform.

Eerste generatie programmeertalen = programmeurs schreven rechtstreeks in binaire code (CPU afhankelijk)

Tweede generatie programmeertalen = Assembler = gebruiksvriendelijker, nadien vertaalt naar binaire code door computer. (geen echte programmeertaal en CPU afhankelijk)

Derde generatie programmeertalen = C, C++, VB, Pascal etc. = gebruiksvriendelijk programmeren in de broncode, nadien omzetten naar objectcode (binair).

### Gecompileerde programmeertalen

Broncode weggeschreven, nadien vertaald naar objectcode (compileren). Iedere CPU heeft een eigen compiler.

Voordelen:

overdraagbaar

snel = reeds omgezet

broncode beschermd

Nadelen:

CPU afhankelijk

besturingssysteem afhankelijk

omslachtig en tijdrovende compilatie

Geïnterpreteerde programmeertalen

Vertalen gedaan tijdens de uitvoering van het programma (GEEN OBJECTCODE).

Voordelen:

snel aanpassen

overdraagbaar

Nadelen:

traag

broncode niet beschermd

Java

Gecompileerd als geïnterpreteerd, broncode .java -> vertalen naar VM machine (bytecode .class) → nadien interpreteren.

Voordelen:

overdraagbaar

snel

gecomprimeerd, handtekening toevoegen

broncode beschermd

CPU onafhankelijk

platformonafhankelijk

Nadelen:

trager door JIT (Just In Time-Compiler)

JVM nodig (Java Virtual Machine)

Kenmerken:

geïnterpreteerd

overdraagbaar

objectgeoriënteerd

gedistribueerd

robuust

multithreaded

veilig

snel

Bibliotheken die de communicatie met het besturingssysteem bezorgen in Windows heet WIN32-API.

Eclipse is een IDE (Integrated Development Environment).

Sequenties zijn opeenvolgingen die worden uitgevoerd in de volgorde dat ze geschreven zijn.

Karakters in Java worden voorgesteld als 16-bits UNICODE-karakters. Als de 9 MSB’s gelijk zijn aan 0, dan komt de UNICODE-codering overeen met de ASCII-codering. ASCII is een subset van UNICODE.

Escapecodes:

\uGETAL getal in UNICODE-codering

\n nieuwe regel

\r wagenterugkeer

\t tab-toets (tabulator)

\b backspace

\f nieuwe pagina

\’ ‘

\” “

\\ \

Decimaal GETAL

Binair 0BGETAL

0bGETAL

Hexadecimaal 0xGETAL

0XGETAL

Octaal 0GETAL

Voordelen van OOP zijn modulariteit (keurige opdeling - indeling in modules) en afscherming van informatie.

Klassenvariabelen: alles van een bepaalde klasse

Instance-variabelen: 1 concreet object in een klasse

Als 2 strings naar de zelfde geheugenplaats verwijzen zijn het canonieke strings.

***2 STRING VERGELIJKEN ALTIJD MET .EQUALS!! NOOIT MET ==!!***

Formatter codes:

b boolean

c character

d integer

f floating point

s string

e wetenschappelijke notatie

n nieuwe regel

- waarde links uitlijnen

+ getal voorzien van een + of - teken

0 getal aanvullen met nullen (vooraan)

, gebruik scheidingstekens bij duizendtallen

( plaatst negatieve getallen tussen haakjes

Een anonieme array is een array waarvan de lengte bepaald wordt door het aantal ingegeven elementen, niet door [#].

# Codes

Commentaar:

/\* commentaar \*/

/\*\* documentatie \*/

// commentaar

Klasse definiëren:

public class CLASS {

//CODE

}

Main methode definiëren:

public static void main(String[] args) { main -> Ctrl + Spatie

//CODE

}

Tekst op het scherm laten verschijnen:

System.out.print(“TEKST”);

System.out.println(“TEKST”); syso -> Ctrl + Spatie

Variabelen declareren (primitieve datatype):

byte BYTE;

short SHORT;

int INTEGER;

long LONG;

float FLOAT;

double DOUBLE;

boolean BOOLEAN;

char ‘KARAKTER’;

Variabelen declareren (referentietype):

Button button = new Button();

Toetsenbord activeren, gebruiken en afsluiten:

import java.util.Scanner;

Scanner NAAM = new Scanner(System.in);

VARIABELE = NAAM.nextVARIABELE();

keyboard.close();

Keuzes:

if (voorwaarde) {

//CODE

} else {

//CODE

}

Herhaling:

for (initialisatie; voorwaarde; code uit te voeren aan het einde van elke for) {

//CODE

}

while (voorwaarde) {

//CODE

}

do {

//CODE

} while (voorwaarde);

Constanten:

final VARIABELE NAAM = WAARDE;

Switch:

switch(value) {

case 1:

//CODE

break;

case 2:

//CODE

break;

…

…

break;

default:

//CODE

Methode:

public static VARIABELE OF void NAAMMETHODE(VARIABELE a, VARIABELE b) {

//CODE

}

Variabele oproepen -> VARIABELE NAAM = NAAMMETHODE(a, b);

Void oproepen -> NAAMMETHODE(a, b);

Classes:

public class NAAM {

//CODE

}

Variabele oproepen -> VARIABELE NAAM = NAAM.NAAMMETHODE(a, b);

Void oproepen -> NAAM.NAAMMETHODE(a, b);

Typeconversie:

(variabele naar om te zetten)WAARDE

Objecten maken:

KLASSE NAAM = new KLASSE();

Garbage collection starten (poging tot!):

System.gc();

Formatter:

Formatter formatter = new Formatter();

String tekst = “Hier komt een getal achter %d”;

formatter.format(tekst,20);

System.out.println(formatter.toString());

formatter.close();

Array maken:

DATATYPE[] NAAM = new DATATYPE[lengte];

DATATYPE[] NAAM = {#, #, #, #};

For each lus:

for (DATATYPE variabele : ARRAYNAAM) {

// CODE

}

Arrays van Arrays maken:

DATATYPE[] [] NAAM = new DATATYPE [lengteRijen] [lengteKolommen];

DATATYPE[] [] NAAM = {{#, #}, {#, #, #}};

Methode met variabel aantal parameters:

public static DATATYPE naam(int… waardes) {

// CODE

}