

Java

Een inleiding tot het programmeren in Java



Inhoud

Inleiding.....	3
Compileren en de kracht van Java	4
Netbeans	4
Opdrachten	4
Probleemanalyse & Jackson.....	5
Algemeen	5
Voorbeeld 1.....	6
Opdrachten Probleemanalyse	7
1. Hallo Wereld!	12
Compileren.....	12
Uitvoeren	12
Bespreking.....	13
Opdrachten Hoofdstuk 1	13
2. Variabelen	14
Primitieve variabelentypes	15
Opdrachten Hoofdstuk 2	15
3. Operator.....	16
Betekenis operatoren	16
Auto-increment operatoren	17
Opdrachten Hoofdstuk 3	17
4. Statements.....	18
If/else statements	18
Voorbeeld.....	19
Opdrachten Hoofdstuk 4	19
5. Lussen.....	20
For-lus	20
While-lus	20
Opdrachten Hoofdstuk 5	20
6. Invoer vanaf het toetsenbord	22
Opdrachten Hoofdstuk 6	23
Algemene opdrachten	24

Inleiding

Deze reader is een afgeleide van het Wikibook: Programmeren in Java, voor meer informatie zie de website van Wikibooks – <http://wikibooks.nl>

Aanvullende bronnen:

<http://thenewboston.org> – Youtube films over diverse onderwerpen, waaronder Java

<http://docs.oracle.com/javase/tutorial/> - Officiële documentatie van Oracle

Object-georiënteerd programmeren

Om een computer te "programmeren", d.w.z. verschillende instructies achter elkaar zetten in een "programma", moest men vroeger veel werk verrichten. Een computer werkt namelijk binair, d.w.z. met enen en nullen. Om een programma te schrijven moest men dus een heel lange serie van enen en nullen achter elkaar zetten. Dit is echter een zeer tijdrovende bezigheid, en het gevolg was dat er niet veel programma's waren en dat het schrijven van een nieuw programma zeer veel tijd in beslag nam. Daarom heeft men de "programmeertaal" uitgevonden, die programmeren een stuk eenvoudiger maakt.

Men probeert programmeertalen zo eenvoudig mogelijk te maken, en tegelijk ervoor te zorgen dat ze een zo groot mogelijk rendement hebben. Dit doet men door analogieën met de werkelijkheid in een programmeertaal te steken. Een programma moet namelijk niet alleen goed draaien, maar ook eenvoudig te verstaan zijn. Daarom heeft men het Object-georiënteerd (OO) programmeren uitgevonden.

Het centrale concept bij OO is het Object. Om een voorbeeld te geven van een Object: kijk even rond u heen. Wat ziet u? Een pen, een bureau, een stuk papier, ... Dit zijn allemaal "objecten". Op eenzelfde manier worden objecten in een OO-taal gebruikt.

Bovendien kunnen Objecten eigenschappen "overerven". Om het principe van overerving te demonstreren, kijkt u even naar een fruitschaal. Wat ziet u? Appelen, peren, sinaasappelen, ... Al deze "objecten" kunnen onder de noemer "Fruit" vallen. Fruit zelf valt dan weer onder de noemer "Voedsel". Voedsel heeft één groot kenmerk: het kan gegeten worden. Fruit erft deze eigenschap over. Een Appel erft dan ook deze eigenschap over, en het gevolg is dat u de Appel kan eten. Een soortgelijk principe wordt toegepast in OO. Het moederobject, waarvan alle objecten overerven, heet dan toevallig of niet ook Object.

Objecten in OO kunnen vrij realistisch zijn, zoals in het voorbeeld met het fruit, maar ze kunnen ook abstracter worden, zoals bijvoorbeeld een lijstje, een stukje tekst, een bestand, een open netwerkverbinding, ... Wanneer u objecten maakt, probeer dan altijd de analogie met de werkelijkheid in het achterhoofd te houden.

Het is onmogelijk om objecten uit het niets te grijpen. Zo is dat ook in een OO-taal. Om objecten te maken, moet u eerst een soort blauwdruk schrijven voor dat object, een "klasse". Dit is wat u doet in de programmeertaal, in dit geval Java. In deze klasse definieert u wat een bepaald object kan en doet, en dan kunnen er eindeloos veel objecten aangemaakt worden. Om terug te komen op het voorbeeld met het fruit: als u eenmaal weet hoe u een appel moet "maken", kan u er eindeloos veel "maken", met verschillende groottes, kleuren en zelfs andere smaken! Dit is de kracht van object-georiënteerd programmeren.

Compileren en de kracht van Java

Zoals eerder gezegd, een computer verstaat alleen enen en nullen. Dus hoe kan een computer ooit een programma uitvoeren in een programmeertaal met complexe dingen zoals objecten? Het antwoord is vrij simpel: voordat je iets met een programma kan doen, moet de code eerst worden omgezet naar enen en nullen. Dit proces heet "compileren".

Bij een "klassieke" programmeertaal, bijvoorbeeld C++, wordt de programmacode omgezet naar code die direct door een computer te begrijpen is. Dat is uiteraard gemakkelijk, want er is geen extra programma nodig om de code uit te voeren. Maar dit betekent wel dat code die voor een systeem is geschreven (bv. Microsoft Windows) niet werkt op andere systemen (bv. Apple Mac OS of Linux). Er moeten dus aparte versies voor die systemen gecompileerd worden, en vaak dient de programmacode eerst worden aangepast omdat deze systemen anders werken.

Met Java is het de bedoeling om platformonafhankelijke programma's te maken. Dat wil zeggen dat een programma dat je thuis maakt in Java zal draaien op jouw besturingssysteem, maar ook op andere besturingssystemen als Linux of MAC OS X, ...

Java lost dit probleem op door gebruik te maken van een compiler die weliswaar een binair bestand aanmaakt, maar waarin de opdrachten nog steeds platformonafhankelijk zijn. In plaats van de broncode rechtstreeks om te zetten naar machinetaal, zoals bij C of C++ e.d. het geval is. Dit heeft tot gevolg dat de Java-compiler geen "stand-alone" uitvoerbaar bestand (een .exe onder Microsoft Windows) maakt. In de plaats daarvan compileert de Java-compiler een bestand in bytecode. Dit bestand wordt dan ingelezen door een "interpreter" ("Java Virtual Machine", JVM) dat op elk platform waar Java programma's op moeten draaien geïnstalleerd moet worden. De JVM leest de bytecode, interpreteert de opdrachten en voert dan het programma uit.

Java-programma → bytecode → machinetaal

De JVM is wel degelijk afhankelijk van het platform, maar is beschikbaar voor veel verschillende systemen, is gratis en hoeft slechts eenmalig te worden geïnstalleerd.

Het uitvoeren van bytecode gaat weliswaar iets trager dan het doorsnee 'native' programma, oftewel een programma dat specifiek voor één systeem is geschreven (onder Microsoft Windows een .exe), maar dit is in het dagelijks gebruik nauwelijks merkbaar. Het voordeel is echter dat je niet afhankelijk bent van beslissingen van de softwareontwikkelaar om een programma voor jouw systeem al dan niet te ontwikkelen.

Netbeans

Om te programmeren in Java gaan we gebruik maken van Netbeans. Netbeans is geprogrammeerd in Java, hierdoor heb je meteen alle benodigde bestanden.

Je kan Netbeans downloaden vanaf: <http://netbeans.org>, let er wel op dat je het volledige pakket download en installeert (eventuele eerdere / andere versies kan je beter eerst deïnstalleren).

Mocht je geen gebruik willen maken van Netbeans kan je ook de "Java Development Kit" downloaden bij Oracle.

Opdrachten

Sla de opdrachten goed op, zodat je ze later aan de docent kan laten zien.

Tip: Sla per hoofdstuk de opdrachten op in een map en geef de opdrachten de naam van de opgave.

Probleemanalyse & Jackson

De Jackson Structured Programming is een methode om gestructureerd te programmeren die beschreven werd door de Engelse informaticus Michael A. Jackson. Aan de hand van de gegevensstroom wordt een programma opgebouwd. Bij de voorstelling hiervan wordt gebruikgemaakt van blokken.

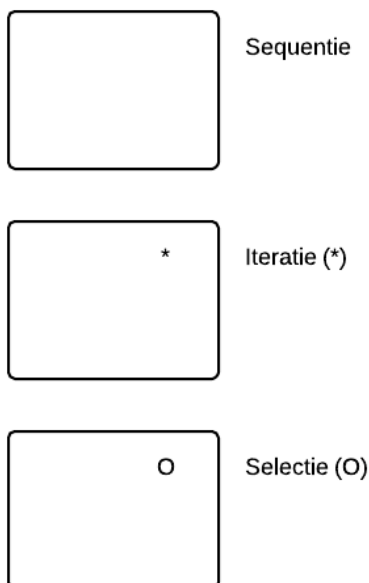
Algemeen

Gegevens of processen zijn op te splitsen in drie elementaire delen:

1. Sequentie op (vaste) volgorde, een voor een, achter elkaar te verwerken.
2. Iteratie herhaling, komt (0 of) meer keer voor; aantal keer ligt keihard vast of is voorwaardelijk vastgelegd.
3. Selectie meervoudige keuze, waarbij een en niet meer dan een blok doorlopen wordt.

Deze elementaire blokken kunnen worden samengevoegd tot complexe totalen, ze blijven echter als blok aanwezig.

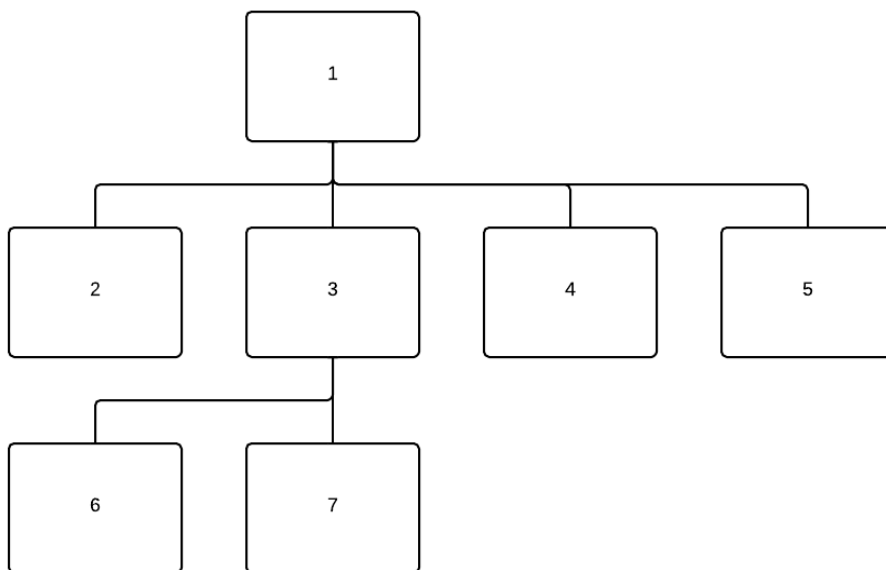
In diagram volgens Jackson:



Een selectieblok komt altijd meer dan een keer voor, naast elkaar. Er moet dan een en niet meer dan een blok gekozen worden.

In samenstellingen dient van links naar recht (sequentieel) gebouwd te worden; om hoofdzaken van bijzaken te helpen onderscheiden kunnen blokken van boven naar beneden nader worden gedefinieerd (een blok bestaat uit onderliggende blokken)

Voorbeeld 1



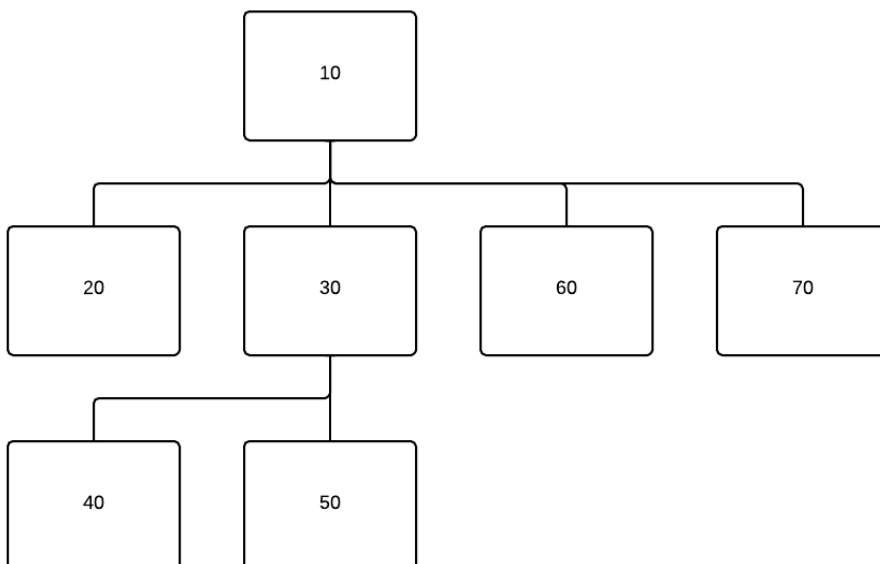
Te interpretern als:

Er was eens een hoop ellende (1),
deze hoop bestaat uit 2, 3, 4 en 5 (in die volgorde)
blok 3 is nog verder onder te verdelen in 6 gevolgd door 7

Praktisch hebben we dus te maken met (achtereenvolgens de blokken) 2, 6, 7, 4 en 5.

Blok 3 is een verzamelnaam voor 6 en 7, blok 1 is een verzamelnaam voor het geheel.

Doordat de volgorde waarin de blokken doorlopen worden in voorgaand voorbeeld niet de normale volgorde is (1, 2, 3, 4, etc.) wordt de nummering herzien tot:



- Alles is zo keurig op volgorde van doorlopen genummerd.
- Door met 10-tallen (of 2-tallen, of 5-tallen, of ...) te nummeren kan in een later stadium nog verder worden gespecificeerd (blok 60 bestaat uit blokken 62, 64, 66, etc).
- De blokken bevatten zelf verder geen informatie; alleen de structuur van het totaal wordt in het diagram weergegeven.
- Alle overige informatie wordt geleverd in vorm van een genummerde lijst:

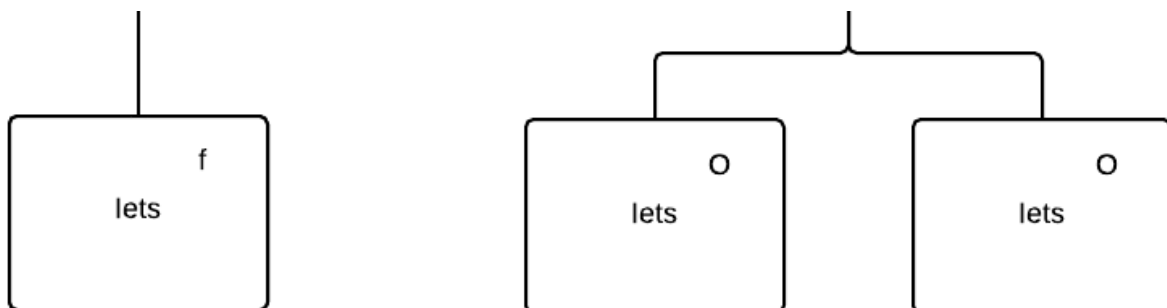
10	Hoop ellende
20	Begin van alle ellende
30	Ellende deel 2
40	Deel 2a
50	Deel 2b
60	Nog meer ellende
70	Laatste druppel

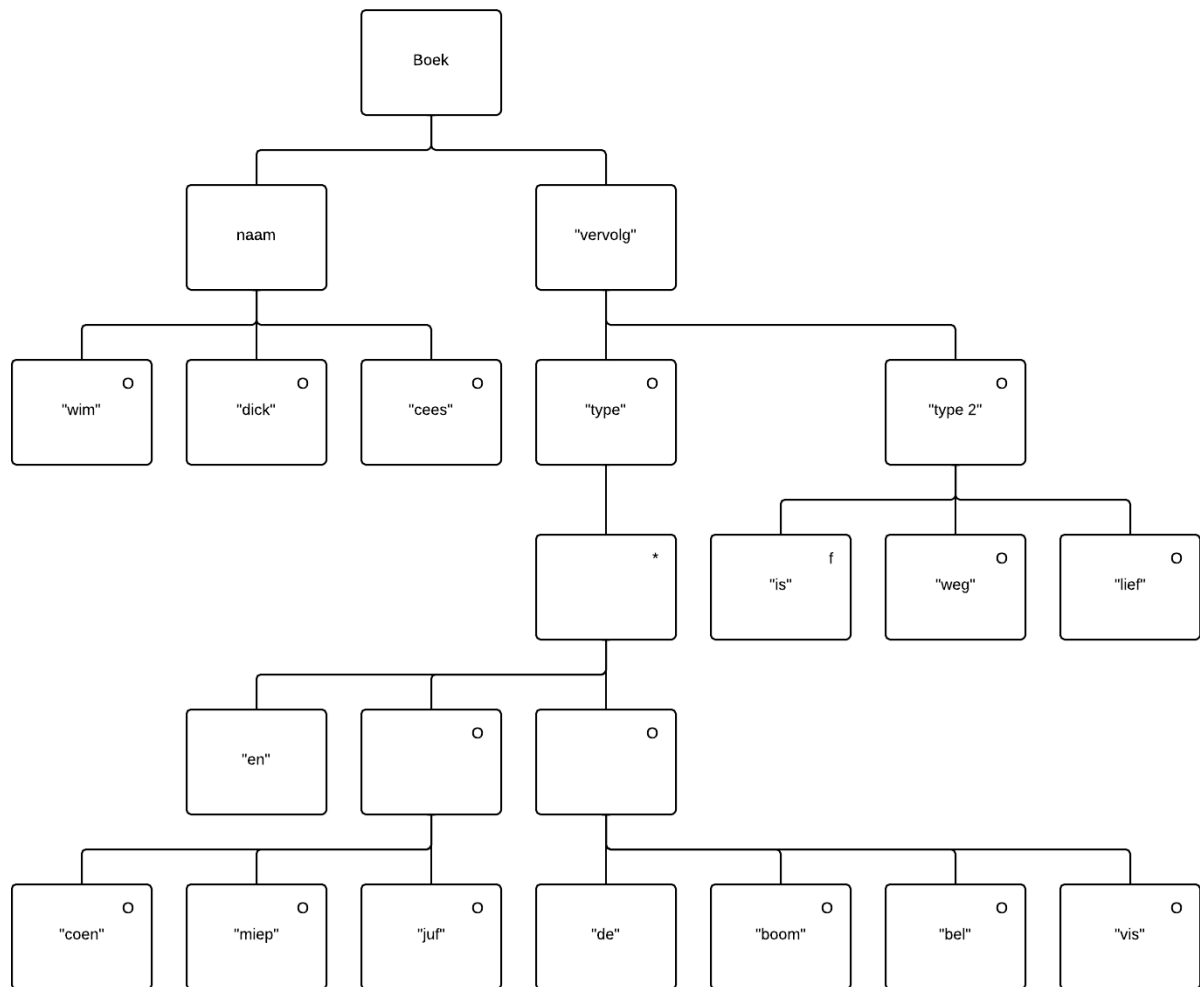
- De zogenaamde action list beschrijft dus alle blokken; blokken 10 en 30 worden vaak niet verder benoemd omdat ze op een lager niveau verder uitgewerkt zijn.

Opdrachten Probleemanalyse

Opdracht 1; Het boek

- In dit diagram zijn alle voorwaarden naar eigen inzicht invulbaar.
- De 'echte' gegevens staan tussen quotetekens ("")
- Een "f" staat voor facultatief, eigenlijk zou hier een of/of situatie moeten staan waarbij een blok leeg is:

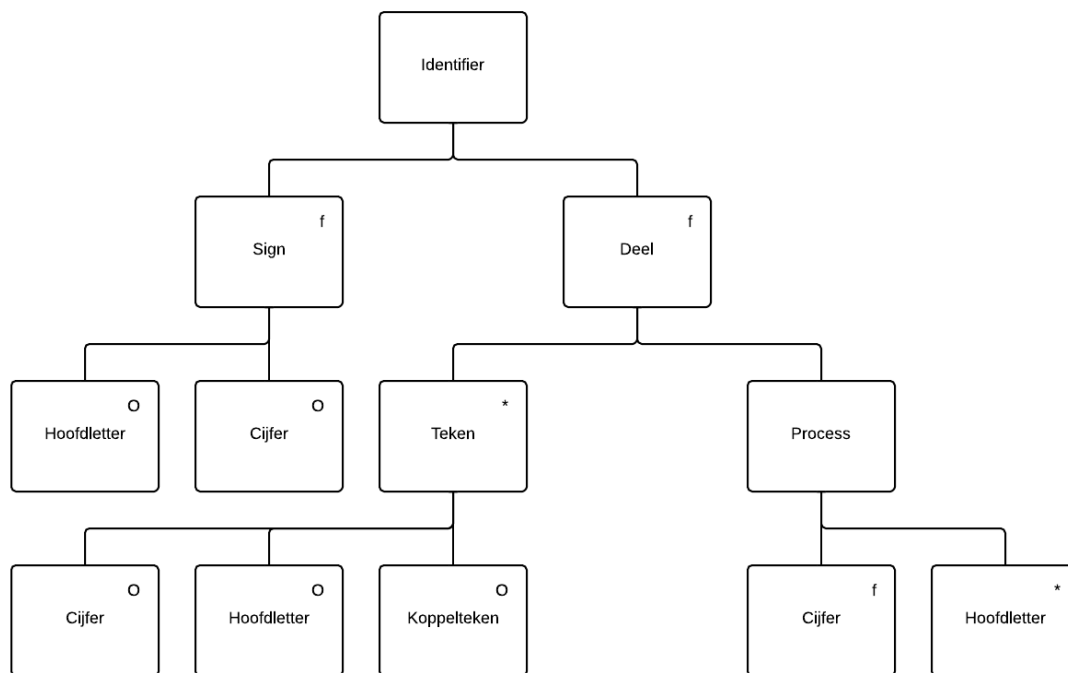




Gevraagd: Welke van de onderstaande zinnen kunnen in het hierboven gedefinieerde boek voorkomen. Geef aan waarom een zin eventueel **niet** kan voorkomen.

1. wim is weg
2. dick weg
3. cees
4. dick cees is weg
5. wim en coen
6. coen en cees
7. cees en juf is weg
8. juf is lief
9. dick en miep cees en juf
10. cees en miep en juf
11. wim coen en juf
12. dick en coen en de vis
13. wim en coen en juf en miep
14. dick en miep en de bel en miep
15. cees en de vis en de bel en juf

Opdracht 2; Identifiers



Identifiers zijn zelf gekozen namen die door een analist / programmeur worden toegekend aan databasevelden of geheugenvariabelen.

Bovenstaande structuur geeft aan hoe identifiers in COBOL zijn opgebouwd.

Elke identifier moet bovendien tenminste een hoofdletter bevatten.

Cijfer is gedefinieerd als: 0, 1, 2, 3, 4, 5, 6, 7, 8 of 9

Hoofdletter is gedefinieerd als: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y of Z

Koppelteken is gedefinieerd als: -

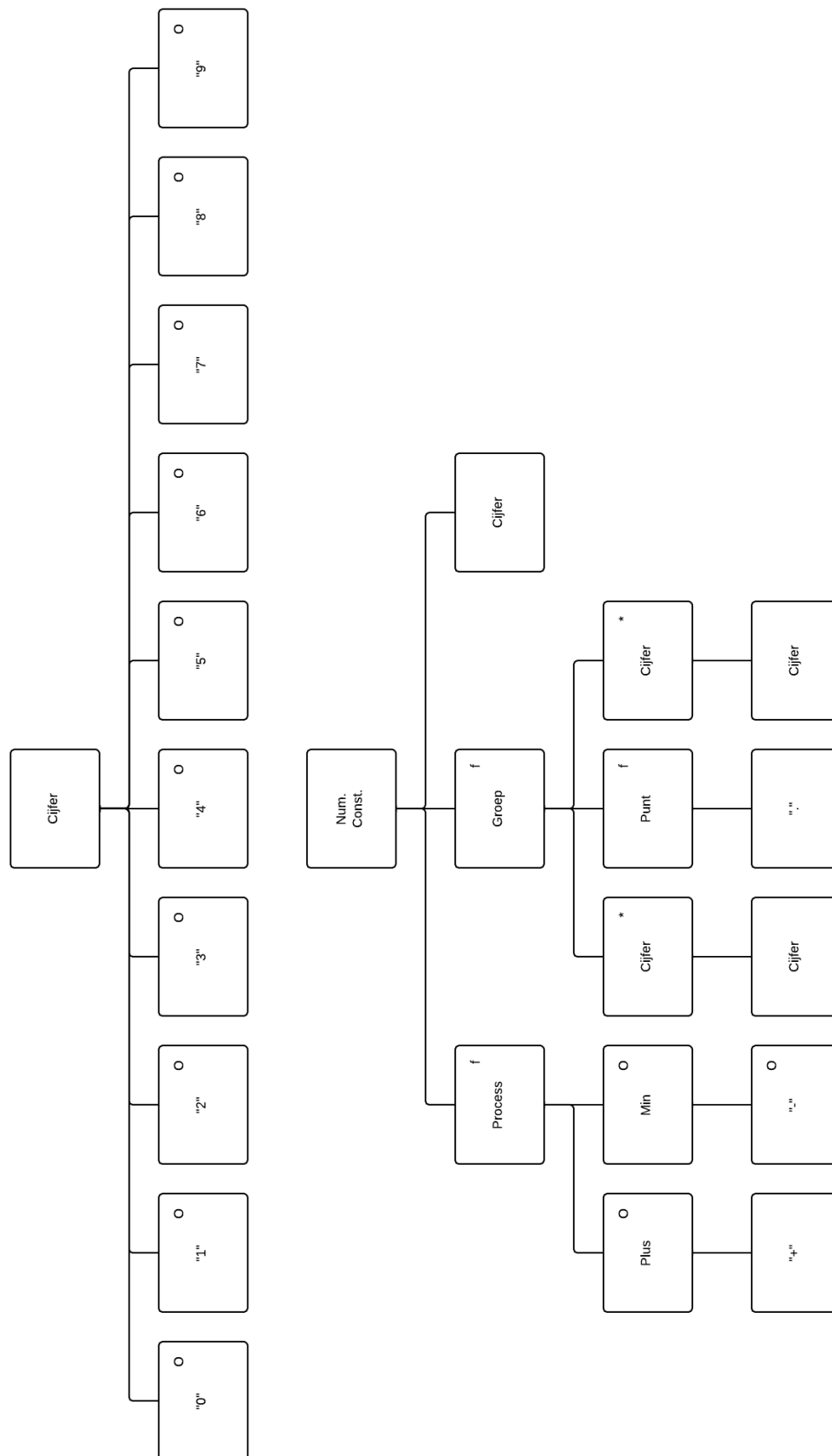
Welke van de onderstaande identifiers zijn foutief en waarom?

1. TOTAAL
2. TO-TAAL
3. A-N-S
4. T---L
5. A128
6. 128A
7. 128
8. -AMC
9. AF
10. NIET GOED
11. WIL.LEM
12. 17%BTW
13. DICK-
14. Cees

n.b.: Alle in het diagram vermelde namen zijn metasymbolen, symbolen (namen) die elders nader gedefinieerd zijn (doorverwijzing naar eerdere definitie).

0, 1, 2, 3, A, B, - zijn eindtermen of terminal symbols.

Opdracht 3; Numerieke constanten



Bovenstaande structuur heeft aan hoe een numerieke constante in COBOL is opgebouwd.

Welke van onderstaande numerieke constanten zijn foutief en waarom?

1. 1.4
2. .0
3. 12.34.56
4. 12 34 56
5. -.45
6. .+69
7. -1284,53
8. -63
9. +1332.645
10. -0.

Opdracht 4; Trein

Probleembeschrijving:

In elke goederentrein is een bewakingswagon opgenomen. Er kunnen twee locomotieven opgenomen zijn, of slechts een. Indien er slechts een locomotief aanwezig is, bevindt deze zich aan het begin van de trein; indien er twee aanwezig zijn bevindt een locomotief zich aan het begin en de andere zich aan het einde van de trein (na de wagons), behalve wanneer er een eindlocomotief is, in welk geval de bewakingswagon zich onmiddellijk voor deze locomotief bevindt. Sommige wagons zijn gekoeld, andere niet.

Gevraagd: Teken een structuurdiagram van de trein

Opdracht 5; Bioscoopavond

Probleembeschrijving:

Een bioscoopavond is nooit te versmaden. Na het journaal en de diverse aankondigingen van films de komende veertien dagen te verwachten zijn, verschijnt als de hoofdfilm een 'korte' is nog een tekenfilm voor de pauze. Na de pauze wordt dan de hoofdfilm vertoond. Als de hoofdfilm een 'lange' is komt de tekenfilm voor de pauze te vervallen en wordt tijdens de hoofdfilm een pauze ingelast.

Gevraagd: Teken een structuurdiagram van de bioscoopavond

1. Hallo Wereld!

We gaan nu ons eerste Java programma schrijven. Daarna gaan we het programma compileren.

Start Netbeans op en maak een nieuw project: **FILE -> NEW PROJECT -> JAVA -> JAVA APPLICATION**

Neem de onderstaande code over, let hierbij wel op dat Java hoofdletter gevoelig is. Sla het bestand vervolgens op als: Hallo.java

```
public class Hallo {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

Compileren

Wanneer je vanuit een commandolijn-omgeving (DOS-prompt, UNIX-shell, Windows: Start > Run > typ "cmd" ...) werkt, ga je nu naar de map waar het bestand Hallo.java staat. Daar typ je het volgende commando:

```
javac Hallo.java
```

Vanuit Netbeans kan je heel eenvoudig op de groene **PLAY** knop drukken, het resultaat wordt onder je code weergegeven.

Dit zou normaal geen fouten mogen geven (als je compiler goed geïnstalleerd werd). In de map zou nu een bestand met de naam Hallo.class moeten staan. Dit bestand is de bytecode van ons "Hallo Wereld"-programma. "javac" is de java-compiler.

Uitvoeren

Om de bytecode uit te voeren gebruik je het programma "java". Dit programma leest de bytecode in en voert deze uit:

```
java Hallo
```

De .class-extensie laat je dus weg. Dit zou als resultaat "Hallo wereld!" in de console moeten afdrukken.

Bespreking

```
1:    public class Hallo {  
2:        public static void main(String[] args) {  
3:            System.out.println("Hello World!");  
4:        }  
5:    }
```

Op regel 1 maken we een nieuwe klasse aan. De public duidt erop dat deze code in het bestand moet zitten met dezelfde naam als de klasse, gevolgd door .java. In dit geval dus Hallo.java. Meer over klassen vind je in het hoofdstuk Klassen.

Op regel 2 zie je de signatuur voor de main-methode. Dit is de instapmethode, je programma begint altijd met deze methode (later meer over methodes).

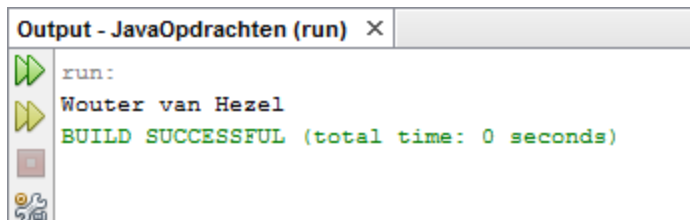
Op regel 3 roepen we een methode op uit de Java API, namelijk de methode println. Deze drukt de meegegeven tekst af in de console. De tekst die meegeeft moet tussen haakjes en tussen aanhalingstekens. In plaats van tekst kan je ook variabelen afdrukken. Daarvoor moet je de naam van de variabele die je wilt afdrukken tussen de haakjes zetten, zonder aanhalingstekens.

Regel 4 en 5 sluiten respectievelijk de main methode en de klasse Hallo terug af. Dit is verplicht. Tip: Het is een goede gewoonte om wanneer je het begin van bijvoorbeeld een methode schrijft ook onmiddellijk de afsluitende accolade te schrijven. Als je een IDE gebruikt, is dit echter niet nodig.

Opdrachten Hoofdstuk 1

Opdracht 1; Druk je volledige naam af op het scherm

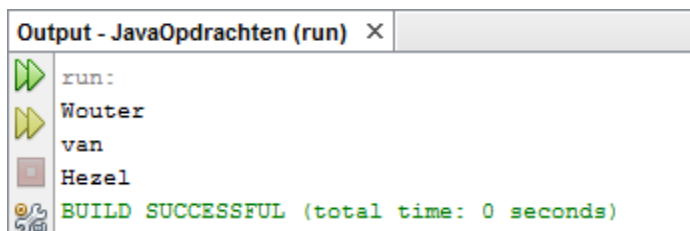
Voorbeeld:



```
Output - JavaOpdrachten (run) X  
run:  
Wouter van Hezel  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Opdracht 2; Druk je volledige naam, gescheiden door enters af op het scherm.

Voorbeeld:



```
Output - JavaOpdrachten (run) X  
run:  
Wouter  
van  
Hezel  
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Variabelen

Variabelen kunnen we gebruiken om tijdelijk gegevens in op te slaan. Deze gegevens kunnen bijvoorbeeld bestaan uit getallen (integers) of stukken tekst (strings), maar kunnen bijvoorbeeld ook waar/niet waar gegevens (booleans) of een reeks gegevens (array) bevatten. Eigenlijk alles wat we in PHP kunnen aanmaken, kunnen we in een variabele zetten.

Nu willen we toch graag wat meer doen dan enkel een tekst weergeven. We zullen ons eerste programma een beetje opfleuren met een wiskundige berekeningen:

```
1:    public class Hallo {  
2:        public static void main(String[] args) {  
3:            int a;  
4:            int b;  
5:            int c;  
6:            a = 1;  
7:            b = 2;  
8:            c = a + b;  
9:            System.out.println("1 + 2 = " + c);  
10:        }  
11:    }
```

Op regel 3 tot 5 maken we drie variabelen aan met respectievelijk de namen a, b en c. Dit doen we via het sleutelwoord `int`, wat de afkorting is voor integer (je mag echter nooit integer voluit schrijven bij declaratie van een variabele). Dit sleutelwoord bepaalt welk soort variabele we aanmaken.

Het aanmaken van een variabele noemen we declareren. Het vertelt de compiler dat we een variabele gaan gebruiken met een bepaalde naam en van een welbepaald type. Het declareren van variabelen is in Java verplicht. Als dat niet gebeurt, dan levert dat een foutmelding op tijdens het compileren.

We kunnen nog op een andere manier een reeks variabelen van hetzelfde type declareren door regel 3 tot 5 te schrijven als:

```
int a, b, c;
```

Deze laatste manier zorgt voor iets compactere code, maar is verder identiek.

Op regel 6 en 7 geven we zowel de variabele a als b een waarde. Men mag niet eender wat na het is gelijk aantekenen zetten (zie verder). De actie die we ondernemen heet initialiseren omdat we voor de eerste keer de variabelen een waarde toekennen. Het is gelijk aantekenen noemen we de toekenningsoperator en lees je best als "wordt" (en niet als "is").

Op regel 8 initialiseren we de variabele c. Niet met een door ons bepaalde waarde maar met de som van de twee variabelen a en b. Het plusteken is een wiskundige operator.

Op regel 9 tonen we een woord op het scherm zoals we eerder al deden. Dat woord is "1 + 2 = ". Daarna schrijven we een plusteken. Dit keer is de plus geen wiskundige operator maar een speciale

"concatenatie"-operator die voor het aaneenzetten van tekst wordt gebruikt (zie het hoofdstuk over Stringbewerkingen).

In Java zitten een paar vaste sleutelwoorden zoals int die bepaalde types van variabelen bevatten. Variabelen die niet meer doen dan één bepaalde waarde opslaan noemen we primitieve variabelen. Deze zijn door Java gedefinieerd en de programmeur kan er geen nieuwe bijmaken. Er zijn in totaal negen primitieve variabelen. Iedere soort staat voor een eigen type. Hieronder zie je alle primitieve typen:

Primitieve variabelentypes

Sleutelwoord	Betekenis	Bits	Bereik	Voorbeeld
boolean	booleaanse waarde	1	true of false	boolean a = true;
byte	heel klein geheel getal	8	-128 (-2^7) tot 127 ($2^7 - 1$)	byte b = 8;
char	karakter	16	Alle Unicode-tekens	char c = 'a';
short	klein geheel getal	16	-32768 (-2^{15}) tot 32767 ($2^{15} - 1$)	short d = 658;
int	geheel getal	32	-2147483648 (-2^{31}) tot 2147483647 ($2^{31} - 1$)	int e = 2000000;
long	groot geheel getal	64	-2^{63} tot $2^{63} - 1$	long f = 220000000;
float	reëel getal	32	$\pm 0,14 * 10^{-64}$ tot $\pm 0,34 * 10^{39}$	float g = 89.567;
double	reëel getal (dubbele precisie)	64	$\pm 0,49 * 10^{-325}$ tot $\pm 0,18 * 10^{309}$	double h = 1000.987;
void	niets	-	-	void methode() { }
string	verzameling karakters			string groet = "Hallo!1"

Opdrachten Hoofdstuk 2

Bij deze opdrachten moet je de gegevens in de juiste type variabele stoppen en afdrukken op het scherm.

Opdracht 3; Java Rules!

Opdracht 4; -127

Opdracht 5; 8,99882

Opdracht 6; G3tal

3. Operator

In java laat een operator je toe (wiskundige) bewerkingen uit te voeren met variabelen. We onderscheiden drie grote groepen operatoren: wiskundige (+, -, *, /, %), logische (&&, ||, !) en relationele (>, <, ==). Daarnaast hebben String-objecten nog een speciale "+" operator ter beschikking waarmee je verschillende Strings tot één String kunt "concateneren" (samenvoegen).

De meeste operatoren doen bewerkingen op twee variabelen, dit noemen we binaire operatoren. Er zijn er ook die op één variabele werken, de zogenaamde unaire operatoren. Er zijn slechts vier unaire operatoren: "+" en "-" als toestandsteken en "++" en "--", waarover later meer.

We zijn al drie (binaire) operatoren tegengekomen: De speciale String "+" operator, de wiskundige "+" operator en de "=" operator die een waarde toekent aan een variabele. Hieronder volgt een lijstje van alle operatoren en een voorbeeld van hun gebruik. Let erop dat je meestal verschillende operatoren tegelijk gaat gebruiken (bv. `a = b + c;`).

Hier onder vind je een greep uit de meest gebruikte operatoren van Java en hun betekenis.

Betekenis operatoren

Symbol	Betekenis	Voorbeeld
=	Een waarde toekennen aan een variabele	<code>a = 5;</code>
De variabele a (eerder gedeclareerd) krijgt waarde 5		
+	Waarden/Variabelen bij elkaar optellen	<code>b = a + 1;</code>
De variabele b krijgt de waarde van (a + 1), hier dus 6		
-	Waarden/Variabelen van elkaar aftrekken	<code>c = b - a;</code>
c krijgt de waarde van het verschil tussen b en a, hier dus 1		
*	Vermenigvuldigen	<code>d = c * 5;</code>
d krijgt de waarde van c vermenigvuldigd met 5, hier dus 5		
/	Delen of	<code>e = d / 10;</code>
e krijgt de waarde van d gedeeld door 10, hier dus 0.5		
==	"Is linkerlid gelijk aan rechterlid?"	<code>boolean m = (5 == 6);</code>
m zal false zijn omdat 5 niet gelijk is aan zes, de == geeft een boolean als resultaat (true of false).		
!=	"Is linkerlid verschillend aan rechterlid?"	<code>boolean n = (5 != 6);</code>
n zal true zijn.		
>	"Is linkerlid groter dan rechterlid?"	<code>boolean o = (5 > 6);</code>
o zal false zijn.		
>=	"Is linkerlid groter dan of gelijk aan rechterlid?"	<code>boolean p = (4 >= 4);</code>
p zal true zijn.		
<	"Is linkerlid kleiner dan rechterlid?"	<code>boolean q = (5 < 6);</code>
q zal true zijn.		
<=	"Is linkerlid kleiner dan of gelijk aan rechterlid?"	<code>boolean r = (6 <= 5);</code>
r zal false zijn.		

Auto-increment operatoren

Je hebt ook nog de auto-increment- en auto-decrementoperatoren. Dit zijn operatoren die je kan gebruiken om een variabele met 1 te verhogen of te verlagen. Ze zien er zo uit: ++ en --. Je kan ze voor of na een variabele zetten (zonder spatie gescheiden) om deze variabele te bewerken (Bvb. "i++;" zal de variabele i met 1 verhogen, "j--;" zal j met 1 verlagen). Bemerkt het verschil in gedrag wanneer de operator voor (preincrementie) of na (postincrementie) de variabele staat.

Voorbeelden

```
public class IncrementKort {  
    public static void main(String[] args) {  
        int a = 1, b = 2, c, d;  
        c = ++a;      // a = 2, c = 2  
        c = a++;      // a = 3, c = 2 (!)  
        d = --b;      // b = 1, d = 1  
        --b;          // b = 0  
        ++b;          // b = 1  
        b++;          // b = 2  
    }  
}
```

Opdrachten Hoofdstuk 3

Bij deze opdrachten moet je de gegevens in de juiste type variabele stoppen en het resultaat van de bewerking afdrukken op het scherm.

Opdracht 7; 8 keer 9

Opdracht 8; 99383 + 884848

Opdracht 9; 11 met 1 verhogen

Opdracht 10; 5595 delen door 998

Opdracht 11; 2999299929292 vermenigvuldigen met 4,2

Opdracht 12; 12 minus 66

4. Statements

Nu we een beetje weten hoe we met variabelen in Java kunnen werken, is het tijd om iets verder te gaan kijken. Alles dat we tot nu toe gezien hebben, zouden we in principe ook nog met simpele code kunnen bereiken. Laten we nu eens wat dieper ingaan op een van de dynamische aspecten van Java.

If/else statements

Statement is het Engels voor 'voorwaarde' en wordt zeer veel in Java gebruikt. Met Java is het mogelijk om verschillende acties uit te voeren afhankelijk van de uitkomst van een bepaalde voorwaarde. De meest simpele voorwaarde die we kunnen stellen is de vergelijking van twee waarden. Zo zou je Java bijvoorbeeld kunnen laten bepalen of een variabele groter of kleiner is dan 5 en afhankelijk daarvan een reactie geven.

Het gebruik van if gaat als volgt:

```
if( <boolean> ) { ... }
```

Tussen de accolades komt de code die moet worden uitgevoerd als de boolean true is. De boolean kan je verkrijgen uit het resultaat van bijvoorbeeld de == operator. Je kijkt of het resultaat van de bewerking true is en voert vervolgens een gewenste bewerking uit. Zo zal code onder "if(4 < 5) { ... }" altijd uitgevoerd worden (daar 4 altijd kleiner is dan 5).

Let erop dat tussen de accolades een willekeurige hoeveelheid programmacode kan komen (net zoals de accolades bij een methode). Tussen de accolades vind je namelijk een code block. Dat is een blok waarbinnen je code kan groeperen. Als het if-statement true evalueert zal hij de volledige blok code tussen de accolades uitvoeren.

Na een if kan je else gebruiken om de code te bepalen die enkel wordt uitgevoerd indien de boolean false was:

```
if( <boolean> ) { ... }  
else { ... }
```

Je ziet dat er bij else geen nieuwe boolean gevraagd wordt. Een else-statement komt altijd onmiddellijk na een if-statement. Tenzij je gebruik maakt van "else if" op de volgende manier:

```
if( <boolean> ) { ... }  
else if( <boolean2> ) { ... }  
else { ... }
```

Voorbeeld

```
1:    public class AbsoluutVerschil {
2:        public static void main(String[] args) {
3:            int a = 19, b = 9;
4:            int verschil = 0;
5:
6:            if(a > b) {
7:                verschil = a - b;
8:            } else {
9:                verschil = b - a;
10:           }
11:
12:           System.out.println("Verschil tussen " + a + " en " + b + " is: " + verschil);
13:       }
14:   }
```

In bovenstaand programma kan zowel a groter zijn dan b of b groter dan a. Het zal niets echter niets uitmaken voor het verschil.

Op regel 6 zien we de vergelijking $a > b$. We maken gebruik van de "groter dan" relationele operator die als resultaat een boolean heeft en dus geschikt is om tussen haakjes van if te staan.

Op regel 7 zien we wat er moet gebeuren indien a groter is dan b. Dan trekken we b van a af zodat het resultaat positief blijft. Indien b groter is dan a moet het omgekeerde gebeuren. Dat zien gebeurt op regel 8 en 9.

Op regel 8 zien we dat de sluitende accolade van het if-statement op dezelfde regel staat als het begin van het else-statement. Dit is een kwestie van stijl, je kan de else ook een regel eronder beginnen. De java compiler is daar blind voor.

Op regel 12 schrijven we de waarde van de twee variabelen en hun onderling verschil uit.

Opdrachten Hoofdstuk 4

Opdracht 13; Vul een variabele met een getal, mocht dit getal groter zijn dan 10, druk dan een foutmelding af op het scherm.

Opdracht 14; Vul een variabele met een getal, mocht dit getal kleiner of gelijk zijn als 5,5, druk dan een melding af op het scherm dat het behaalde cijfer een onvoldoende is. In het andere getal is er een voldoende behaald.

Opdracht 15; Combineer de bovenstaande opdrachten tot een geheel. Let hierop dat de invoer alleen geldig is van 0 tot en met 10.

5. Lussen

Een andere constructie waarbij het dynamische aspect van Java naar voren komt, is de lus. Dit is een constructie waarbij een bepaald stuk code herhaaldelijk uitgevoerd wordt tot aan een bepaald statement wordt voldoen. Er zijn verschillende soorten lussen die we kunnen gebruiken. Ik zal de twee die het meest gebruikt worden, de while-lus en de for-lus hier bespreken.

For-lus

Met de for-lus kun je een gecontroleerde herhaling laten plaats vinden.

```
public class ForTest {
    public static void main (String args[]) {
        for (int i=0; i < 5; i++ ) {
            if (i==0) System.out.println ("i heeft de waarde "+i);
            if (i > 0) System.out.println("plus 1 is "+i);
            if (i == 4) System.out.println("Aan het einde van deze lus heeft i de
waarde "+i);
        }
    }
}
```

While-lus

Zolang aan *statement* voldaan wordt, zal de code in de loop, die wederom tussen accolades {} staat, telkens opnieuw uitgevoerd worden. Vaak wordt een while-loop in combinatie met een tellertje gebruikt, bijvoorbeeld om de getallen 1-5 weer te geven:

```
public class WhileTest {
    public static void main(String[] args) {
        int i=0;
        System.out.println("i heeft de waarde "+i);
        while (i<5) {
            i++;
            System.out.println("plus 1 is "+i);
        }
        System.out.println("Aan het einde van deze lus heeft i de waarde "+i);
    }
}
```

Opdrachten Hoofdstuk 5

Opdracht 16; Druk de getallen 1 t/m 5 onder elkaar af op het scherm met behulp van een for-lus

Opdracht 17; Druk de getallen 10 t/m 15 achter elkaar af op het scherm met een while-lus.

Opdracht 18; Maak de volgende figuren na met behulp van lussen:

Figuur 1	Figuur 2	Figuur 3
* * * * *	* ** *** **** *****	***** **** *** ** *
Figuur 4	Figuur 5	Figuur 6
11111 22222 33333 44444 55555	1 22 333 4444 55555	11111 2222 333 44 5

Extra opdrachten lussen (voor diegene die van een uitdaging houden):

Figuur 7	Figuur 8	Figuur 9
* ** *** **** *****	1 12 123 1234 123456 1234567 12345678	* *** ***** ***** ***** ***** *****
Figuur 10	Figuur 11	
1 121 12321 1234321 123454321 12345654321 1234567654321 123456787654321	1 232 34543 4567654 567898765 67890109876 7890123210987 890123454321098	

6. Invoer vanaf het toetsenbord

Voor het verwerken van gegevens is het handig als je de gebruiker om gegevens kan vragen. Vanzelfsprekend kan dit ook met Java. Zie onderstaand voorbeeld:

```
// Importeer extra bibliotheken voor de BufferedReader
import java.io.*;

public class invoer {
    public static void main(String[] args) throws IOException {
        // Maak een nieuwe instantie aan van BufferedReader (om de invoer te lezen)
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

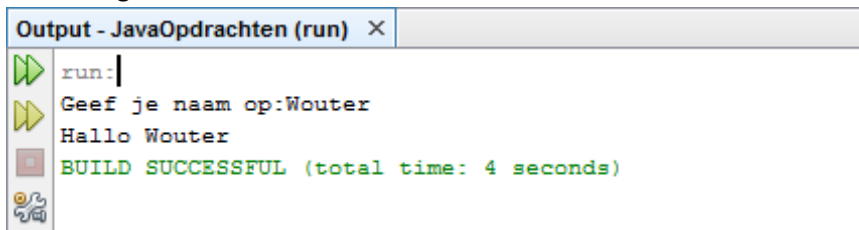
        // Maak een nieuwe variabele aan van het type string:
        String invoer;

        System.out.print("Geef je naam op:");

        // De variable invoer vullen:
        invoer = br.readLine();

        // Uitvoer naar het beeldscherm:
        System.out.println("Hallo " + invoer);
    }
}
```

Uitwerking:



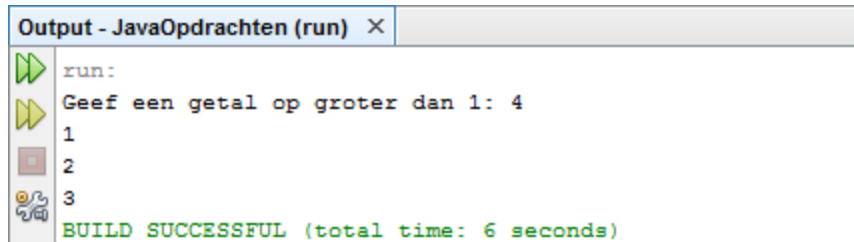
```
run:|
Geef je naam op:Wouter
Hallo Wouter
BUILD SUCCESSFUL (total time: 4 seconds)
```

Opdrachten Hoofdstuk 6

Opdracht 19; Vraag de gebruiker om zijn naam in te voeren, druk daarna zijn naam af op het scherm.

Opdracht 20; Vraag de gebruiker om een getal groter dan 1, druk de voorgaande getallen af op het scherm.

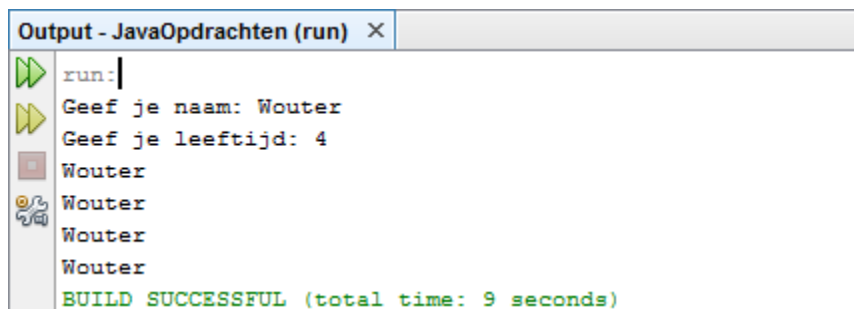
Voorbeeld:



```
Output - JavaOpdrachten (run) X
run:
Geef een getal op groter dan 1: 4
1
2
3
BUILD SUCCESSFUL (total time: 6 seconds)
```

Opdracht 21; Vraag de gebruiker om zijn naam en leeftijd, druk hierna zijn naam zoveel keer als zijn leeftijd af op het scherm.

Voorbeeld:



```
Output - JavaOpdrachten (run) X
run:
Geef je naam: Wouter
Geef je leeftijd: 4
Wouter
Wouter
Wouter
Wouter
BUILD SUCCESSFUL (total time: 9 seconds)
```

Algemene opdrachten

Opdracht 1; Reeks van Fibonacci

Probleemomschrijving:

De reeks van Fibonacci wordt gevormd door een (oneindig) aantal natuurlijke getallen. Het eerste getal is 0, het tweede getal is 1; voor elk volgend getal geldt dat het de som is van zijn twee voorgangers.

Gevraagd; Druk de eerste 20 getallen van de reeks van Fibonacci af op het scherm.

Opdracht 2; Faculteit

Probleemomschrijving:

De faculteit van een getal is gedefinieerd voor alle natuurlijke getallen.

De faculteit van 0 is 1, de faculteit van 1 is ook 1.

De faculteit van 5, geschreven als 5! is $5 * 4 * 3 * 2 * 1$.

Gevraagd; Vraag de gebruiker om een getal, druk daarna de faculteit en de uitkomst af op het scherm.

Opdracht 3; Machtsverheffen

Probleemomschrijving:

3⁵. 3 tot de macht 5 is: $3 * 3 * 3 * 3 * 3$, dus 243

24. 2 tot de macht 4 is: $2 * 2 * 2 * 2$, dus 16

Gevraagd; Vraag de gebruiker om een getal en de macht, druk daarna de uitwerking en de uitkomst af op het scherm.

(Extra) Opdracht 4; Binair – Decimaal conversie

Probleemomschrijving:

De gebruiker voert getallen in; enen en nullen (binair)

Gevraagd; Bereken het decimale getal