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’s uitvoeren 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 resulteren 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.

# Object georiënteerd programmeren

Een **object** heeft **attributen**/**eigenschappen**/**variabelen** en **methoden**. Een programma bestaat meestal uit verschillende objecten die onderling boodschappen naar elkaar sturen, deze boodschappen noemen methoden.

** Definitie**: Een **object** is een groepering van eigenschappen (attributen) en gerelateerde codeblokken (methoden).

richting = auto.draaien(hoek);

resultaat = bestemmeling.methode(parameter);

Een **klasse** is een blauwdruk die de eigenschappen en methoden van objecten van dezelfde soort bepaalt.

Een **object** is een unieke instantie van een klasse. Bij de new operator en de naam van de klasse wordt een geheugen gereserveerd voor het object, het resultaat is een verwijzing (reference) naar het object.

Klassenaam variabelenaam = new Klassenaam(parameters);

Random generator = new Random();

Random generator; //declaratie  
generator = new Random(); //initialisatie

In dit vb. is generator een verwijzing naar een Random object, een object kan ook meerdere referenties hebben.

Random generator2;

generator2 = generator;

## Pakket/package

Elke **klasse** behoort tot een **package**.

Random behoort tot java.util (java.util.Random). Om Random met de korte naam (Random i.p.v. java.util.Random) te gebruiken heb je een import statement nodig.

import java.util.Random //Random klasse  
import java.util.\* //alle klassen uit de package java.util

De compiler zoekt naar de juiste klasse via:

1. expliciete imports
2. klassen huidige package
3. klassen uit geïmporteerde pakketten

Automatische imports voor:

* default package (klasse zonder pakket)
* huidige package (klassen in hetzelfde pakket)
* package java.lang (basisklassen van Java)

## Random

Klasse om willekeurige getallen te genereren met 2 constructors:

Random() //telkens andere getallen  
Random(long seed) //telkens dezelfde opeenvolging van getallen (seed)

 Gebruik de Random constructor slechts éénmaal in je programma!

.nextInt(int n) //willekeurig geheel getal van 0 tot waarde van n  
.nextDouble() //willekeurig getal van 0 tot 1  
.nextBoolean() //willekeurige boolean (true of false)

## Objecten gebruiken

Van een **object** kan je alleen de **publieke** **attributen**(variabelen) en **methoden** gebruiken.

Random random = **new** Random();

int dobbelsteen = random.nextInt(6) + 1; //om 0 te vermijden (1 tem 6)  
boolean muntstuk = random.nextBoolean();  
int muntstuk = random.nextInt(); //waarden 0 en 1

## Geheugen

**Garbage** **collection** zorgt voor het opruimen van objecten waar geen enkele referentie meer naar is. Dit als de referentie niet meer bestaat (out of scope) of als de waarde null aan de referentiewaarde wordt toegekend.

**** Dit gebeurt automatisch maar nooit onmiddellijk (afzonderlijke thread met lage prioriteit).

## String

**String** is een **klasse** (object behorende tot java.lang) en is immutable wat wil zeggen dat de inhoud niet meer gewijzigd kan worden na creatie. Elke verandering resulteert in de creatie van een nieuw String object. Bestaande String literals worden slechts éénmaal in het geheugen geplaatst.

Strings kunnen via een Scanner object ingelezen worden via .next() (tot spatie of enter) of via .nextLine() (tot enter).

String tekst = **new** String(**“Hello Shizzle”**);  
String tekst = **“Hello Shizzle”**;

 Vergelijk strings nooit met ==

## StringBuilder

**StringBuilder** is een alternatief voor String dat wel **mutable** (aanpasbaar) is: chars toevoegen, tussenvoegen, verwijderen, ... zonder telkens een nieuw object te moeten maken. Daardoor is dit veel sneller en zuiniger met geheugenruimte als er veel manipulaties aan een String moeten gebeuren. Enkel te creëren met constructor.

StringBuilder emptyBuilder = **new** StringBuilder();  
StringBuilder emptyBuilder = **new** StringBuilder(**“For Shizzle”**);

## Gegevens formatteren

System.*out*.printf(**“%d appels kosten €%.2f.\n”**, aantal, prijs);  
//is hetzelfde als  
String str = String.format(**“%d appels kosten €%.2f.”**, aantal, prijs);

|  |  |
| --- | --- |
| **%d** | voor **int** en **long** |
| **%f** | voor (**float**) en **double** |
| **%s** | voor **String** |
| **%c** | voor **char** |
| **%b** | voor **boolean** |
| **%n** | voor een newline (beter alternatief dan **\n** – platform onafhankelijk) |
|  |  |
| **n** | totaal minimum aantal posities (bv. **%20s**) |
| **-** | links uitlijnen (bv. **%-20s**) |
| **n.d** | geeft bij %f het totaal aantal posities en aantal decimalen met afronding (bv. **%6.2f**) |

**** Voor elke formaataanduiding moet er een met het type overeenstemmende variabele of literal aanwezig zijn.

System.*out*.printf(**“De %dde letter is %d\n”**, 5, ‘e’); //de 5de letter is e

# Klassen

Om een programma te kunnen uitvoeren is er altijd één klasse nodig met een main methode.

**** Een klasse dewelke een main methode bevat moet altijd public zijn!

public static void main(String[] args) {}

public [abstract | final] class Classname extends Superclass implements Interface {}

De structuur van een klasse ziet er als volgt uit:

package graphics;

public class Rectangle { //klassedeclaratie of class header

//klasseomschrijving of class body  
  
 private int x; //attributen private omwille van encapsulation  
 private int y; // enkel datatype en naam zijn verplicht  
 private int height;  
 private int width;

public Rectangle(int width, int height) { //constructor  
 this.width = width;  
 this.height = height;  
 }

public int getHeight() { //methode(s)  
 return height;  
 }

public void setHeight(int height) {  
 this.height = height;  
 }  
}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | klasse zelf | package klassen | subklassen | alle klassen |
| **private** | x |  |  |  |
| ***package-private*** | x | x |  |  |
| **protected** | x | x | x |  |
| **public** | x | x | x | x |

## Scope van variabelen

Een **member** **variabele** (attribuut) wordt bovenaan de klasse gedeclareerd, krijgt een beginwaarde en is geldig in heel de klasse.

Een **lokale** **variabele** wordt gedeclareerd binnen een codeblok, krijgt geen beginwaarde, is alleen geldig binnen het codeblok en mag eventueel dezelfde naam hebben als een attribuut.

public class Voorbeeld {  
 private int waarde; //member variabele (attribuut)

public void eenMethode() {  
 int waarde = 1; //lokale variabele  
 }  
}

Je kan onderscheid maken tussen een lokale variabele en een attribuut door het this sleutelwoord.

public class Voorbeeld {  
 private int waarde = 42;

public void eenMethode() {  
 int waarde = 7;  
 System.*out*.println(waarde);  
 System.*out*.println(this.waarde);   
 }  
}

## Gegevens doorgeven aan methoden

Je kan gegevens aan een methode doorgeven via **parameters**, binnen de methode te beschouwen als lokale variabelen). Elke parameter dient **apart** gedeclareerd te worden.

public void setPosition(int xpos, int ypos) {  
 this.x = x;  
 this.y = y;  
}

Rectangle rectangle = new Rectangle();  
rectangle.setPosition(7, 9);

In dit voorbeeld wordt de methode aangeroepen met de concrete waarden **7** en **9**. Deze waarden worden ***by value*** doorgegeven aan x en y.

## Getters (accessors)

Deze methoden geven informatie over een attribuut van een object, meestal geen parameters en de return waarde is van hetzelfde type als het attribuut.

String name;

public String getName(){  
 return name;  
}

## Setters (mutators)

Dienen om een eigenschap van een object te wijzigen en hebben meestal één parameter van hetzelfde type als het te wijzigen attribuut. Er is geen returntype, dus void. Vaak controleert men de nieuwe waarde(n).

public void setName(String name){  
 if(!name.isEmpty()){  
 this.name = name.toUppercase();  
 }  
}

## Method overloading

Wanneer je verschillende methoden dezelfde naam geeft kiest de compiler de “juiste” methode op basis van de meegegeven parameters. De parameters moeten verschillen in type en/of aantal.

## Constructor

Iedere klasse heeft een constructor, deze is nodig om een object te maken en te initialiseren. Een constructor heeft nooit een returntype en mag eveneens geen void in de methode signatuur hebben. De naam moet overeenkomen met de naam van de klasse.

Indien geen constructor gedefinieerd genereerd de compiler een default constructor zonder parameters.

public class Television {

public Television(){  
 }

public Television(String brand, int size){  
 this.brand = brand;  
 this.size = size;  
 }  
}

Om vanuit een constructor een andere constructor op te roepen maakt men gebruik van **this()**, dit moet echter altijd als eerste regel in de constructor geschreven worden.

public Television() {  
 this(“Sony”, 43);  
} //via deze constructor roept men de constructor in het vb. hierboven aan

## toString()

Ieder object heef een methode toString die een String object teruggeeft, als je deze methode niet overschrijft in de klasse zal de standaard methode toegepast worden.

Dummy dummy = new Dummy();

System.*out*.println(dummy.toString()); //Dummy@190d11  
System.*out*.println(dummy); //zelfde resultaat (automatisch toString)

Aangeraden is om voor elke klasse zelf een toString methode te schrijven.

public class Rectangle { //klassedeclaratie of class header

//klasseomschrijving of class body  
  
 private int x; //attributen private omwille van encapsulation  
 private int y; // enkel datatype en naam zijn verplicht  
 private int height;  
 private int width;

public Rectangle(int width, int height) { //constructor  
 this.width = width;  
 this.height = height;  
 }

// ...

public String toString(){  
 return “(“ + x + “,” + y + “)”;  
 }

// OF

public String toString(){  
 return String.format(“(%d,%d)”, x, y);  
 }  
}

# Arrays

Een array of reeks is een verzameling van elementen waarbij alle elementen van hetzelfde datatype dienen te zijn. Een array zelf is een object en heeft een vaste lengte (aantal elementen).

## Default waarden

|  |  |
| --- | --- |
| **0** | *gehele getallen* |
| **0.0** | *decimale getallen* |
| **false** | *booleans* |
| **null** | *objecten* |

De creatie van een array kan op verschillende manieren gebeuren.

int[] numbers = {1, 3, 6};  
char[] letters = {‘z’, ‘o’, ‘n’}; //3 elementen na creatie

int[] lottogetallen = new int[45]; //45 elementen na creatie

boolean[] aanwezigheden;  
aanwezigheden = new boolean[24]

System.*out*.println(aanwezigheden.length); //methode length op elke array

## Arrays van objecten

Arrays kunnen ook referentietypen bevatten, deze hebben default null als waarde.

String[] woorden = new String[3];

System.*out*.println(woorden[0]); //null  
System.*out*.println(woorden[1]); //null

woorden[0] = “Aap”;  
System.*out*.println(woorden[0]); //Aap

//declaratie met onmiddellijke initialisatie  
String[] woorden = {“Aap”, “Noot”, “Banaan”};

Je kan van een String ook een array van Strings maken:

String tekst = “Ik heb de zon zien zakken in de zee”;  
String[] woorden = tekst.split(“ “); //split de String op spaties

Van een String een array van karakters (char) maken:

String tekst = “Ik heb de zon zien zakken in de zee”;  
Char[] tekens = tekst.toCharArray();

## Arrays van arrays

Een element van een array kan zelf ook een array zijn, zo kan je een “matrix” declareren met rijen en kolommen.

int [][] cijfers = new int[2][3];

De eerste index geef de rij aan, de tweede index de kolom.

|  |  |  |  |
| --- | --- | --- | --- |
|  | kolom 1 | kolom 2 | kolom 3 |
| rij 1 | cijfers[0][0] | cijfers[0][1] | cijfers[0][2] |
| rij 2 | cijfers[1][0] | cijfers[1][1] | cijfers[1][2] |

int [][] cijfers = {{1, 2 , 3}, {4, 5, 6}};

Elementen uit een array afdrukken kan via een dubbele for-loop of via een dubbel for-each loop.

for (int i = 0; i < cijfers.length; i++){  
 for (int j = 0; j < cijfers[i].length; j++){  
 System.*out*.print(cijfers[i][j] + “ “);  
 }  
 System.*out*.println();  
}

/\* afdruk  
1 2 3  
4 5 6  
\*/

for (int[] rijen : cijfers) {  
 for (cijfer : rij){  
 System.*out*.print(cijfer + “ “);  
 }  
 System.*out*.println();  
}

/\* afdruk  
1 2 3  
4 5 6  
\*/

# Overerving

///DRAFT

2 klassen met dezelfde attributen => 1 superklasse met 2 subklassen waarbij de superklasse de gemeenschappelijke attributen bevat

Square (subklasse) extens Rectangle (superklasse) (klassenhiërarchie), subklasse erft over van/is een soort van superklasse.

een subklasse erft eigenschappen en methoden over van superklasse, kan zelf eigenschappen en methoden toevoegen en kan deze overriden( (één superklasse per klasse/één keer extends), die superklasse kan wel subklasse zijn van een andere superklasse enz

overriden = eigenschap met dezelfde naam en datatype, je krijgt bijgevolg de overschreven eigenschapswaarde

DON’T do this met eigenschappen, wel met methoden (bad practice)

//TOEGANGSNIVEAU DIAGRAM

alle public en protected methoden van de superklasse kunnen we gebruiken vanuit de subklasse, een subklasse is een uitbreiding van de superklasse (alle niet private eigenschappen en methoden van de superklasse zijn ook beschikbaar in de subklasse).

override (overschrijven/verbergen) is wanneer we in een subklasse een andere implementatie geven van een bestaande methode in de superklasse (dezelfde signatuur = naam + parameters + return type)

!elke methode uit een superklasse is standaard overschrijfbaar in een subklasse

public class Square extends Rectangle {  
 private int side;

//met deze annotatie in de subklasse om de compiler te laten nakijken of deze methode wel degelijk een methode uit zijn superklasse overschrijft  
 @Override  
 public int getArea(){  
 return side \* 2;  
 }  
}

om aan te duiden dat de methode uit de superklasse gebruikt moet worden

public void setSide(int side){  
 super.setWidth(side);  
 super.setHeight(side);  
}

Constructors

Indien geen geschreven => default

//default contructor van normale klasse, in dit geval superklasse  
public Rectangle(){  
}

//default constructor van subklasse  
public Square(){  
 super();  
}

Indien een subklasse gebruikt maakt van een constructor van de superklasse, moet deze met de juiste parameters opgeroepen worden in de subklasse

public Square(int side){  
 super(this, this); //altijd eerste statement  
 this.side = side;  
}

SLIDE21