Java spiekbrief

Commentaar

Dient enkel voor de 'menselijke' lezer.

```
// ... tot het einde van de lijn

/* Dit type commentaar kan gerust
   meerdere lijnen beslaan. */

/** En deze speciale vorm wordt
    gebruikt voor programmadocumen-
    tatie (javadoc).
*/
```

Types

`int`	Geheel getal
`double`	Reëel getal
`boolean`	Logische waarde
`char`	Letterteken
`String`	Reeks lettertekens
Klasse	Object van de opgegeven klasse

Variabelendeclaratie

Steeds *type* gevolgd door *naam*:

```
int aantal;
String resultaat;
boolean verkocht;
```

Definitie van een klasse

```
public class Naam {

    // velddefinities
    //    (= variabelendeclaraties)
    ...

    // constructordefinities
    ...

    // methodedefinities
    //    (= functies/procedures)
    ...
}
```

(De exacte volgorde van de onderdelen is niet zo belangrijk.)

Definitie van een veld

Algemene vorm: '`private` *type naam*;'.

```
private double basisloon;
private Persoon chef;
private String naam;
```

Definitie van een methode

```
public ...hoofding... {
   // declaraties van lokale variabelen
   ...

   // opdrachten
   ...
}
```

Drie soorten methodes (verschillende hoofdingen):

- Constructoren
- Procedures
- Functies

```
public Klassennaam (...parameters...)
    // = constructor

public void procedureNaam (...parameters...)

public type functieNaam (...parameters...)
```

Parameterdefinities (variabelendeclaraties) gescheiden door komma's. Ook haakjes als er geen parameters zijn.

```
public Persoon (String naam,
                double leeftijd) {
    ...
}

public void setNaam (String nieuweNaam) {
    ...
}

public double getLeeftijd() {
    ...
}
```

Naamconventies voor *getters* en *setters*.

Namen

- Klassennaam begint met een hoofdletter
- Primitieve types: kleine letters
- Constanten: allemaal hoofdletters
- De rest: kleine letters
- Gebruik 'camel case'

```
class Persoon {...}
class LidVanClub {...}

int aantal;
String naamVanKlant;

public void print (String str) {...}

private static final double
                CM_PER_INCH = 2.54;
```

Namen bestaan uit letters en cijfers en mogen niet beginnen met een cijfer.

Declaratie van lokale variabele

Variabelendeclaratie, vaak met initialisatie.

```
int aantal;

String resultaat = "";
double totaal = 0.0;
double opp = lengte*breedte/2.0;
```

Opgelet! Zonder `private`

Eenvoudige opdrachten

Toewijzing:

```
naam = uitdrukking;
```

Procedure-oproep:

```
procedureNaam (uitdrukking, ...);
object.procedureNaam (uitdrukking, ...);
```

Print-opdracht:

```
System.out.println (uitdrukking);
```

Return-opdracht (enkel binnen functie):

```
return uitdrukking;
```

Elke opdracht eindigt met een puntkomma, ook de laatste!

Uitdrukkingen

Combinaties van termen met bewerkingen:

`+ -`	Optellen, aftrekken
`* /`	Vermenigvuldiging, deling
`%`	Rest bij deling
`+`	Stringconcatenatie

Opgelet! 7/3 geeft 2, 7.0/3.0 geeft 2.333...

Termen zijn

- Naam van een veld, parameter, variabele
- Constanten

  ```
  0 -10 2.5 "Hello there" true false
  ```

- Functie-oproep

  ```
  functieNaam (uitdrukking, ...);
  naam.functieNaam (uitdrukking, ...);
  ```

- Constructor-oproep

  ```
  new Klassennaam (uitdrukking, ...);
  ```

Selectie

```
if (conditie) {
   // opdrachten
   ...
} else if (conditie) {
   // opdrachten
   ...
} else {
   // opdrachten
   ...
}
```

Het `else`-gedeelte mag weggelaten worden.
Het `else if`-gedeelte komt 0 of meer keer voor.

Condities

Uitdrukking met `boolean` waarde (`true` of `false`).

uitdrukking < *uitdrukking* *uitdrukking* > *uitdrukking* *uitdrukking* <= *uitdrukking* *uitdrukking* >= *uitdrukking*	
uitdrukking == *uitdrukking*	(gelijk)
uitdrukking != *uitdrukking*	(verschillend)

Opgelet! Gebruik == en != enkel met primitieve types (int, double, ...). Voor objecten: gebruik functie `equals(...)`.

```
if (aantal == 0) {
   ...
}
if (opdracht.equals("STOP")) {
   ...
}
```

Combinaties van condities:

conditie `&&` *conditie*	EN
conditie `||` *conditie*	OF
`!` *conditie*	NIET

Verhogen / verlagen

Afgekorte opdrachten om aantallen te verhogen of te verlagen

Opdracht	Betekenis
naam ++	*naam* = *naam* + 1
naam --	*naam* = *naam* − 1
naam += *getal*	*naam* = *naam* + *getal*
naam -= *getal*	*naam* = *naam* − *getal*

Lussen

Zolang conditie voldaan is:

```
while (conditie) {
  // opdrachten
  ...
}
```

Lus met teller:

Voert opdrachten uit met $i = 0, \ldots, 9$:

```
for (int i = 0; i < 10; i++) {
    // opdrachten
    ...
}
```

Met $i = 9, 8, \ldots, 1$:

```
for (int i = 9; i >= 0; i--) {
    // opdrachten
    ...
}
```

Met $i = 2, 4, 6, \ldots, 20$:

```
for (int i = 2; i <= 20; i+=2) {
    // opdrachten
    ...
}
```

Arrays (tabellen)

Declaratie

```
int[] tab;
Persoon[] vakgroep;
```

Moet eerst worden gecreëerd

```
tab = new int[5];
Persoon[] vakgroep = new Persoon[10];

String[] namen = new String[aantal];
```

Index van 0 t.e.m. *length*-1.

```
int eerste  = tab[0];   // gebruiken
int laatste = tab[4];

tab[2] = 17;            // toewijzen

// overlopen
for (int i=0; i < namen.length; i++) {
   String naam = namen[i];
   // doe iets met naam
   ...
}
```

Speciale '*for each*'-lus waarmee je alle elementen van een tabel doorloopt van voor naar achter (zonder index):

```
for (String naam : namen) {
  // doe iets met naam
  ...
}
```

Meerdimensionale tabellen

```
double[][] afstand = new double[10][10];
int min = afstand[0][1];
for (int i=0; i < 9; i++) {
  for (int j=i+1; j < 10; j++) {
    if (min < afstand[i][j]) {
      min = afstand[i][j];
    }
  }
}
```

Strings

Individuele letters kunnen niet gewijzigd worden. Strings zijn *onveranderlijk* (Engels: *immutable*.)

`str.length()`	Aantal tekens
`str.charAt(i)`	Teken op positie *i*
`str.indexOf(ch)`	Positie van teken *ch*
`str.substring(i)`	Substring vanaf positie i
`str1 + str2`	Concatenatie
`str1.equals(str2)`	Zelfde inhoud?

Posities worden geteld vanaf 0.

Nog veel meer 'ingebouwde' methoden — zie elektronische documentatie.

Lijsten

Lijken op arrays, maar kunnen groeien en gebruiken een andere notatie. Opgelet! Elementen moeten *objecten* zijn. (Zie ook: wikkelklassen.)

Import nodig (bovenaan bestand)

```
import java.util.ArrayList;
```

of (af te raden): `import java.util.*;`

Declaratie

```
ArrayList<Persoon> vakgroep;
```

Creatie

```
vakgroep = new ArrayList<Persoon> ();
ArrayList<String> namen = new ArrayList<> ();
```

Aantal elementen

```
namen.size()
```

Opvragen met *get*. Index van 0 t.e.m. *length*-1.

```
String eerste  = namen.get(0);
String laatste = namen.get(name.size()-1);
```

Achteraan toevoegen met *add*

```
vakgroep.add (persoon);
```

Verwijderen met *remove* (op index)

```
vakgroep.remove (index);
```

For each-lus:

```
for (Persoon p : vakgroep) {
  // doe iets met p
  ...
}
```

Heeft heel veel 'ingebouwde' methoden — zie elektronische documentatie.

Wikkelklassen (wrapper classes)

Wikkelen primitief type in object.

`int`	`Integer`	`double`	`Double`
`char`	`Character`	`boolean`	`Boolean`

Voornamelijk gebruikt met lijsten

```
ArrayList<int> aantallen;     // mag niet!
ArrayList<Integer> aantallen; // OK
```

Conversie van primitief type naar wikkelklasse is doorgaans automatisch:

```
int aantal = aantallen.get(0);
aantallen.add (rijen*kolommen);
for (int aantal : aantallen) {
  ...
}
```

This

`this` verwijst naar het 'huidige' object.

Vaak gebruikt om onderscheid te maken tussen lokale variabelen (of parameters) en velden:

```
public class Persoon {

  private String naam;

  public Persoon (String naam) {
    this.naam = naam;
  }
}
```

Null

Een variabele die *nergens* naar verwijst, heeft de waarde `null`.

```
if (namen[2] == null) {
  ...
}

persoon = null;

return null;
```

Voorbeelden

- Variabele gedeclareerd, maar niet geïnitialiseerd
- Elementen van nieuw gecreëerde arrays
- Expliciet `null` toegewezen aan de variabele

Object

Methodes die elk object heeft/moet hebben:

Zet object om naar een String:

```
public String toString ()
```

Vergelijk met een ander object:

```
public boolean equals (Object ander)
```

Bepalen van een hash-waarde (voor gevorderden):

```
public int hashCode ()
```

Java voorziet automatisch standaardimplementaties van deze drie methoden.

Klassenvariabelen

- Aangeduid met `static`
- Horen bij de klasse, niet bij het object
- Slechts één instantie per klasse

Voornamelijk gebruikt voor constanten:

```
private static final int MAX = 60;

private static final String[] TALEN = {
   "nl", "fr", "en"
};
```

Klassenmethoden

- Aangeduid met `static`
- Horen bij de klasse, niet bij het object
- Kunnen velden van object niet gebruiken

Vaak gebruikt voor 'rekenkundige' functies.

```
public static double kwadraat (double g) {
    return g*g;
}
```

Worden opgeroepen met klassennaam i.p.v. object.

```
double waarde = Math.cos(x)*Math.sin(x);
```

Uit de Java-bibliotheek:

`Math.cos(x)` `Math.sin(x)`
`Math.tan(x)` `Math.log(x)`
`Math.exp(x)` `Math.sqrt(x)`
... zie klasse *Math*
	Wiskundige functies

`Integer.parseInt(str)` `Double.parseDouble(str)`	String omzetten naar getal

De methode main

Opgeroepen wanneer een klasse wordt *uitgevoerd* als alleenstaand programma.

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

Parameter `args`: opdrachtlijnargumenten

```
c:\Users\kc> java MijnKlasse piet jan
```

Overerving

Klassendefinitie:

```
public Klasse extends Superklasse {
    ...
}
```

- Een klasse kan maar één (directe) superklasse hebben.
- *Object* is (indirecte) superklasse van *alle* klassen

Constructor van *Klasse* gebruikt constructor van *Superklasse* als *super(...)*

```
public Klasse (...) {
    super (...);
    ...
}
```

Methode van *Klasse* kan 'verborgen' methodes gebruiken van *Superklasse* met behulp van '*super.*'

```
public void beweeg (int dx, int dy) {
  ...
  super.beweeg (2*dx, 2*dy);
  ...
  super.beweeg (-dx, -dy);
  ...
}
```

Velden van *Superklasse* alleen toegankelijk voor *Klasse* als ze *protected* zijn (i.p.v. private)

```
protected String naam;
```

Abstracte klassen en methoden

Abstracte *methode*: hoofding zonder implementatie

```
public abstract void voegGewichtToe();
```

Abstracte methoden enkel toegelaten in abstracte *klassen*.

```
public abstract class AbstracteKlasse {
   ...
}
```

Een abstracte klasse mag ook niet-abstracte methoden hebben (mét implementatie).

Definitie van een interface

```
public interface Naam {

  // methodedeclaraties
  //    (= functies/procedures)
  ...
}
```

- Geen velden,
- Geen constructoren,
- Naam begint met hoofdletter zoals bij een klasse

Methodedeclaratie: enkel hoofding, geen corpus.

```
public interface Doos {

  double getGewicht ();

  void voegGewichtToe ();

  boolean nogPlaatsVoor (double gewicht);
}
```

(Alle methoden zijn automatisch publiek)

Implementatie van een interface

Een klasse *Klasse* kan aangeven dat ze een interface *Interface* implementeert:

```
public class Klasse implements Interface {
  ...
}
```

Een klasse kan tegelijk meerdere interfaces implementeren en ook nog een andere klasse uitbreiden

```
public class Klasse
    extends Superklasse
    implements Inter1, Inter2, Inter3 {
  ...
}
```

Een interface kan andere interfaces uitbreiden (maar niet een andere klasse)

```
public interface Interface
    extends Inter1, Inter2, Inter3 {
  ...
}
```