Ian Angillis

1ste Bachelor Toegepaste Informatica

Programming Basics samenvatting

**Hoofdstuk 1: Inleiding**

Zie ppt en cursus

**Hoofdstuk 2: De Java Development Kit**

Zie ppt en cursus

**Hoofdstuk 3: Mijn eerste Java-toepassing**

Zie ppt en cursus

**Hoofdstuk 4: programmatielogica**

Samengevat bestaat een programma doorgaans uit de volgende elementen:

1. Sequenties: een opeenvolging van instructies
2. Keuzes: een voorwaardelijke uitvoering van instructies
3. Herhalingen: een herhaaldelijke uitvoering van instructies
4. Invoer: gegevens opvragen
5. Uitvoer: gegevens ter beschikking stellen

De volgorde en combinatie van deze elementen wordt bepaald door het programmeeralgoritme

**Programmeeralgoritme**: een opeenvolging van instructies die voert tot de oplossing van een probleem.

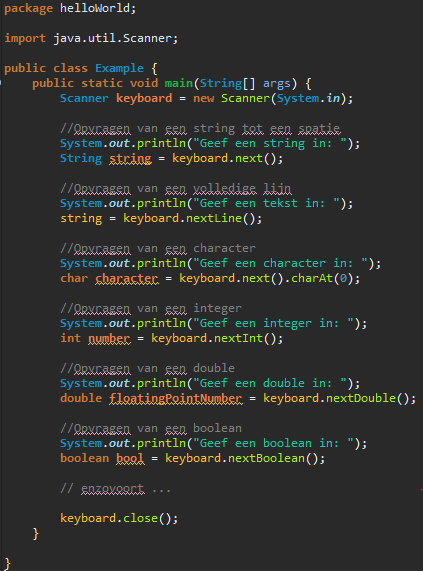
**4.1 Sequenties**

Spreekt voor zich, de computer loopt door de code van uw programma.

**4.2 Invoer en uitvoer**

We kunnen informatie aan de gebruiker opvragen en informatie of het scherm printen

We vragen input aan de gebruiker met een **java.util.Scanner object**.

Een paar voorbeelden: 

**4.3 Keuzes**

De opeenvolging van acties is niet altijd een rechtlijnig pad. Soms moeten er keuzes gemaakt worden en kan het pad verschillende kanten opgaan.

Hiervoor maken we gebruik van if - else statements.

**4.4 Herhalingen**

Soms moeten bepaalde activiteiten herhaaldelijk uitgevoerd worden om tot een resultaat te komen. Dit noemen we iteraties.

Hiervoor maken we gebruik van for, while, do - while en for each loops.

**4.5 Wat wordt er verwacht van de programmeurs**

Programmeurs moeten 2 dingen kunnen:

1. **Het probleem verknippen tot kleine acties of instructies** die de computer voor ons kan uitvoeren. Dus ze moeten gebruik maken van de bovengenoemde elementen
2. **Het samenstellen van de volgorde** waarin deze bovengenoemde acties moeten uitgevoerd worden, met inbegrip van eventuele keuzes en herhalingen.

Een programma is

* een opeenvolging van instructies (sequenties)
* met eventueel keuzes (selecties)
* en herhalingen (iteraties)

om zo tot de oplossing van een probleem te komen

**Hoofdstuk 5: De Java programmeertaal**

**5.1 Inleiding**

Hoofdstuk gaat over:

* De variabelen en hun mogelijke waarden
* de bewerkingen of operatoren
* uitdrukkingen
* programmeerregels
* code-blokken
* instructies voor het verloop van het programma

**5.2 Variabelen en letterlijke waarden**

In een programmeertaal moeten we gebruik kunnen maken van gegevens. Deze gegevens worden tijdelijk opgeslagen in het geheugen zodat de computer er bewerkingen mee kan doen.

**variabele:** een geheugenlocatie met een naam waar je een waarde in kan stoppen.

Elke variabele heeft de volgende kenmerken:

* Type: Vb. int, double, char, .. In Java is iedere variabele van een bepaald type.
* Naam (identifier): Iedere variabele heeft een naam die bestaat uit letters en cijfers. De naam moet steeds beginnen met een letter.
  + lowerCamelCasing
  + Kies zinvolle namen
* Bereik (scope): Een variabele kan maar gebruikt worden binnen een bepaald gebied van het programma. Namelijk het blok waarin ze bestaan.

**5.2.1 De declaratie van variabelen**

Alvorens een variabele te kunnen gebruiken moet hij gedeclareerd worden. We moeten eerst de geheugenruimte in de computer reserveren vooraleer we deze kunnen gebruiken.

Declaratie van variabelen.

<type> <name>;

We geven dus eerst aan van welk datatype de variabele is en vervolgens wat zijn naam is.

Vb. int number; String string; double a; …

Het toekennen van een waarde aan de variabele kan zowel tijdens als na de declaratie gebeuren (initialisatie)

<type> <name> = <value> ;

of

<type> <name>;

<name> = <value>;

**5.2.2 Het datatype**

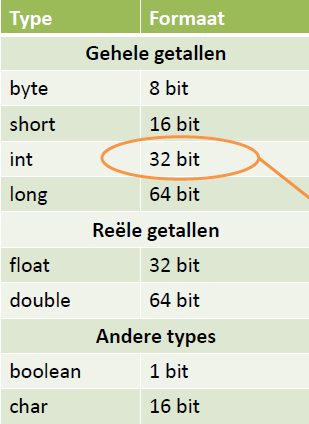
In Java kan men het datatype onderverdelen in twee grote categorieën: het **primitieve datatype** en het **referentietype**.

**5.2.2.1 Het primitieve datatype**

Het primitieve datatype bevat een enkele waarde

Bijvoorbeeld: een getal, een letter of een booleaanse waarde.

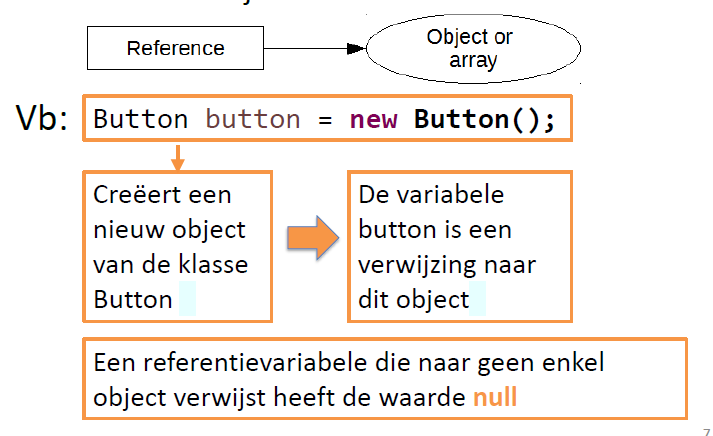
Tabel van primitieve datatypes en hun formaat



Er bestaan geen unsigned integers en dergelijke in Java.

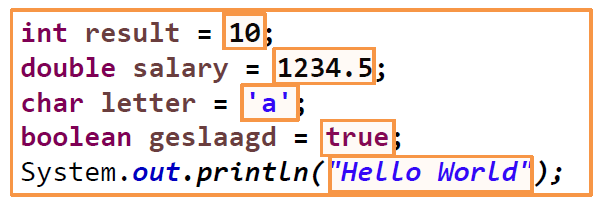
**5.2.2.2 Het referentietype**

Het referentietype is een verwijzing of referentie naar een object. Het referentietype bevat het adres van een object. De inhoud van de variabele is dus eigenlijk de geheugenlocatie van iets anders. (pointer)



**5.2.3 Literals**

Literals zijn letterlijke waarden



floating-point literals zonder de letters D of F zijn standaard van het type double

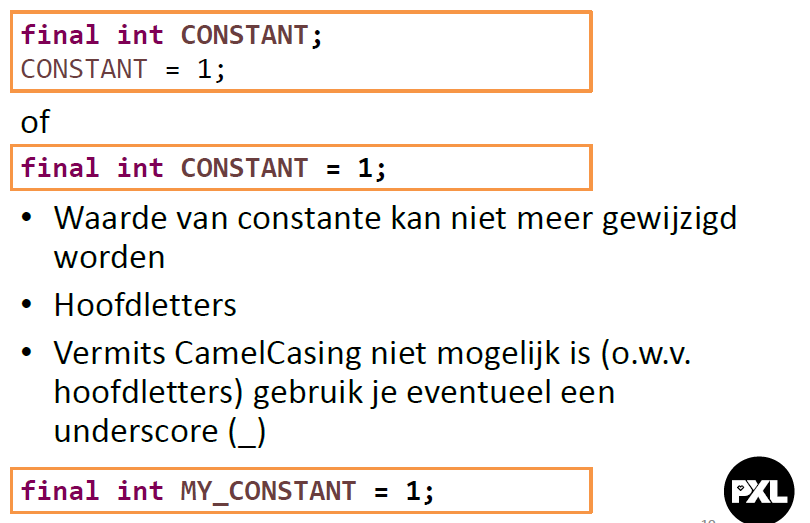
**5.2.4 De naam**

Iedere variabele heeft een naam of identifier die aan de volgende voorwaarden moet voldoen:

* Beginnen met een letter, $ of \_ ($ en \_ worden afgeraden)
* lowerCamelCasing
* Uniek binnen scope
* Mag geen gereserveerd woord zijn

Tabel van gereserveerde woorden in Java:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

**5.2.5 Final variables of constanten**

Final variables zijn variabelen waarvan men de inhoud niet meer kan wijzigen zodra die is toegekend. Declararen van final variables gaat als volgt

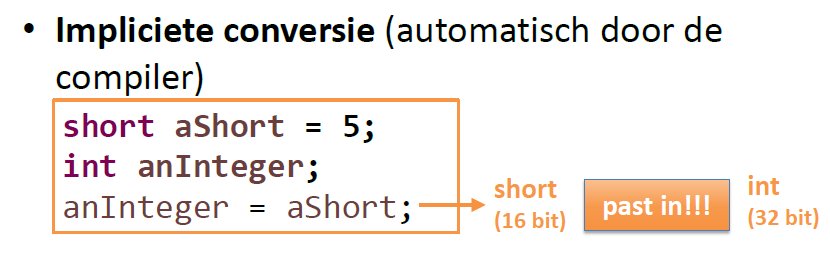
**5.2.6 Typeconversies**

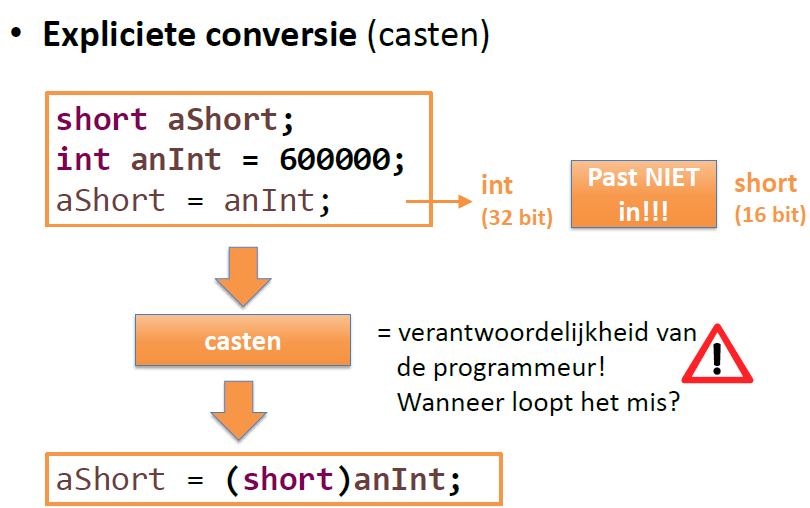
Gegevens van een bepaald type kunnen geforceerd worden naar een ander type. Dit noemt men type casting of typeconversie

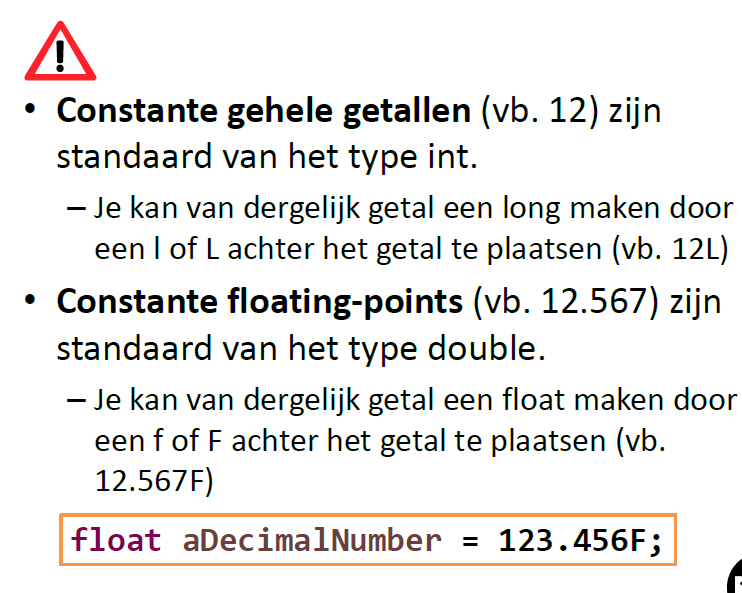
Dus m.a.w, gegevens van een bepaald datatype omzetten naar een ander datatype.

Typeconversie wordt door de compiler meestal automatisch toegepast, zolang er geen gegevens verloren kunnen gaan. Indien er gevaar is voor verlies van gegevens zal de compiler weigeren dit te doen.

Je kan het vergelijken met het overgieten van de inhoud van één doosje in een ander doosje. Als het tweede doosje groter is dan het eerste, kan het overgieten probleemloos verlopen. Als het tweede doosje echter kleiner is, lopen we het risico dat het overloopt en dat er dus gegevens verloren gaan.

Dus er bestaat zowel impliciete als expliciete conversies





**5.3 Operatoren**

Zodra we in het geheugen plaats gereserveerd hebben voor onze gegevens, kunnen we starten met het bewerken van deze gegevens. De basisbewerkingen worden uitgevoerd door middel van operatoren. Een operator neemt een of meerdere waarden en voert er een bewerking op uit. Dit resulteert in een nieuw gegeven waar we dan vervolgens weer iets mee kunnen gaan doen.

We kunnen de operatoren onderverdelen volgens het soort bewerking dat ze uitvoeren

* Rekenkundige operatoren
* relationele operatoren
* logische operatoren
* shift operatoren
* bit operatoren
* toekenningsoperatoren
* andere operatoren

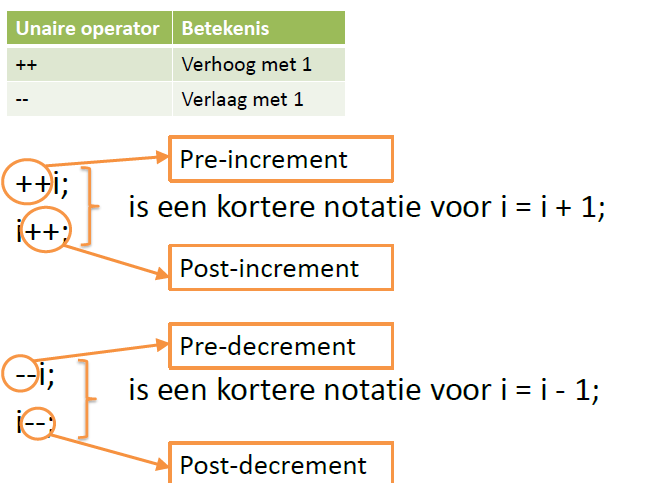
**5.3.1 Rekenkundige operatoren**

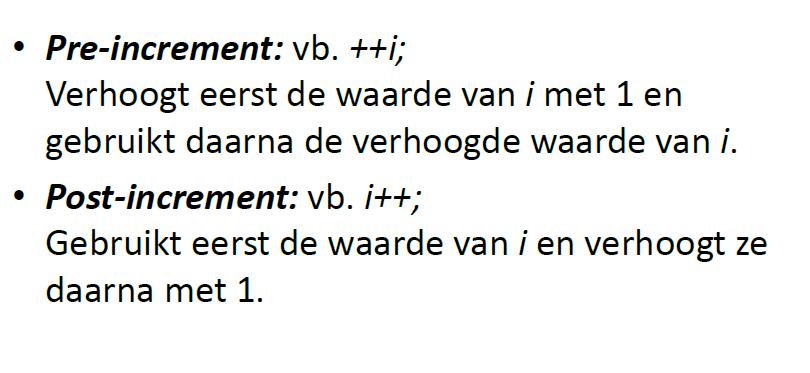
We kunnen operatoren verdelen in 2 soorten

**Unaire operatoren**: deze voeren een bewerking uit op slechts 1 operand

<operand1> <operator>;

<operator> <operand1>;





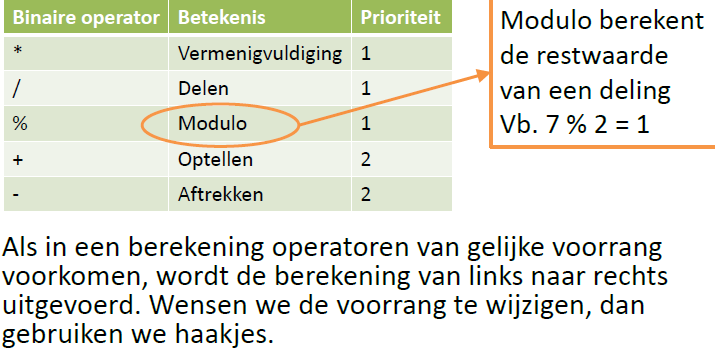
!Opletten!

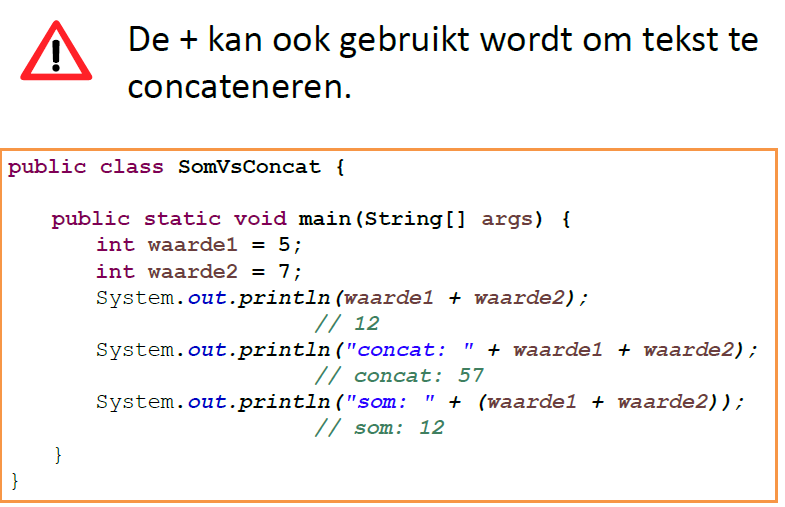
De operatoren + - \* / % hebben geen invloed op de waarden van de operanden zelf. De operatoren ++ en -- wel. Deze kunnen dus ook enkel toegepast worden op variabelen. Niet op letterlijke waarden.

**Binaire operatoren**: deze voeren een bewerking uit op twee operanden

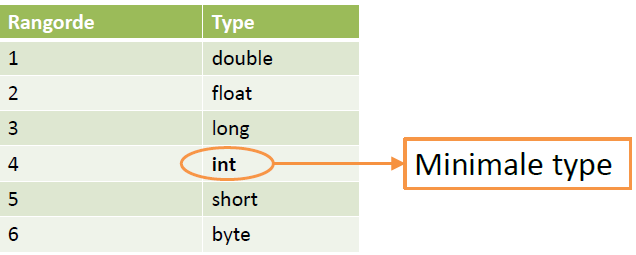
<operand1> <operator> <operand2>

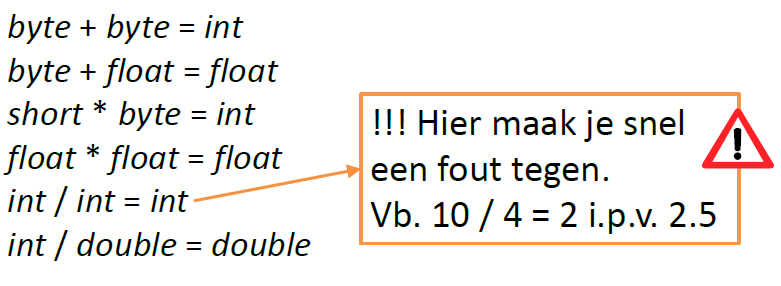
Tabel rekenkundige operatoren:

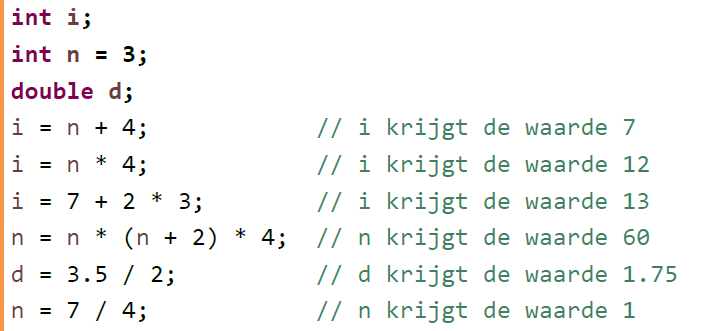




Alle rekenkundige operatoren geven telkens een numerieke waarde als onmiddellijk resultaat. Het datatype van het resultaat is minimaal int, tenzij een van de operanden een type heeft dat hoger is in rang. In dit geval is het uiteindelijke type datgene van de operand met de hoogste rang.







**5.3.2 Relationele operatoren (= vergelijkingsoperatoren)**

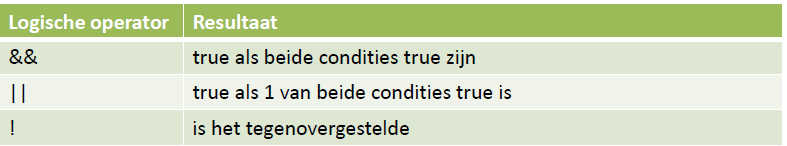
Relationele operatoren bepalen de relatie tussen twee operanden en geven als resultaat de waarde true of false (boolean)

****

**5.3.3 Logische operatoren**

Logische operatoren worden gebruikt om logische combinaties te maken van operanden van het type boolean. Het resultaat is steeds een booleaanse waarde (true of false).

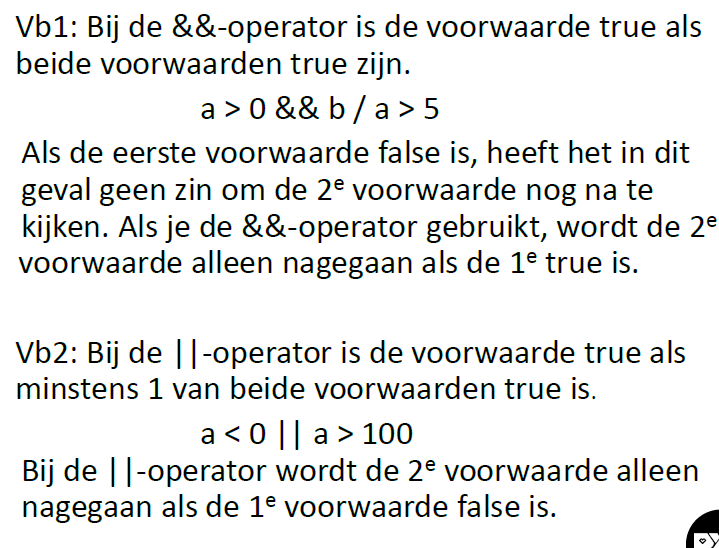
Twee of meer relatiecondities kunnen gecombineerd worden m.b.v. logische operatoren



**Short circuit evaluation**: Indien bij de evaluatie van de 1ste operand het resultaat reeds bepaald is wordt de 2de operand niet meer geëvalueerd.

Dit kan van belang zijn wanneer de 2de operand uit een bewerking bestaat

Voorbeelden:

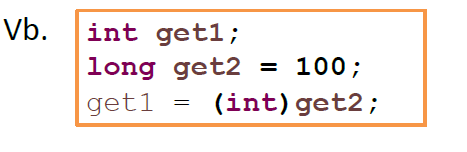


**5.3.4 Toekenningsoperatoren**

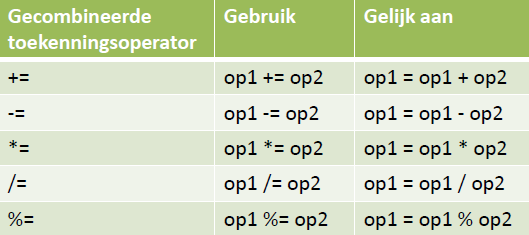
Er is slechts één toekenningsoperator = die de waarde van de tweede operand aan de eerste toekent.



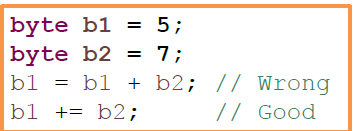
Indien het datatype rechts van het =- teken van een hogere rangorde is, zal de compiler een foutmelding geven. Een expliciete typeconversie (casting) is dan vereist.



De toekenningsoperator wordt vaak gecombineerd met andere operatoren.

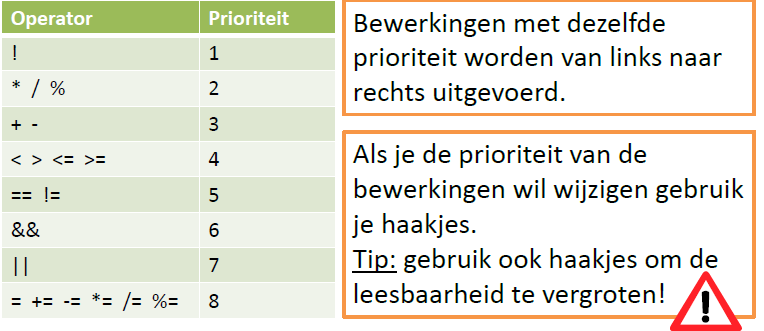


Bij het gebruik van de gecombineerde toekenningsoperator geldt de regel van minimale datatype niet.



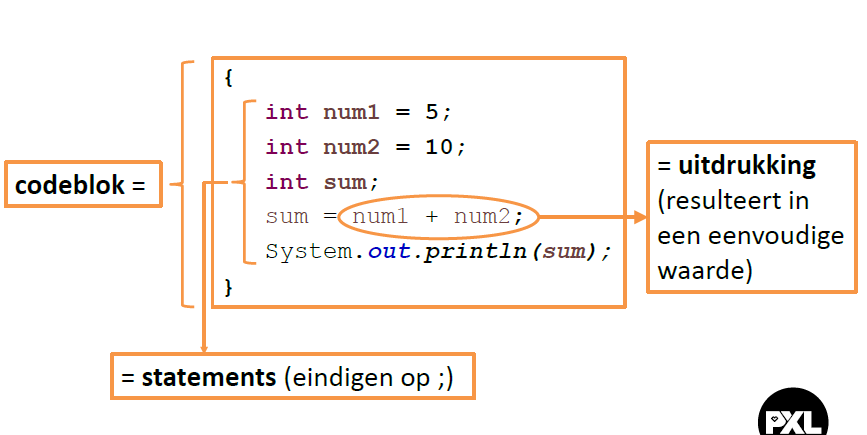
**5.3.7 Prioriteitsregels**

Het is mogelijk meerdere operatoren te combineren

****

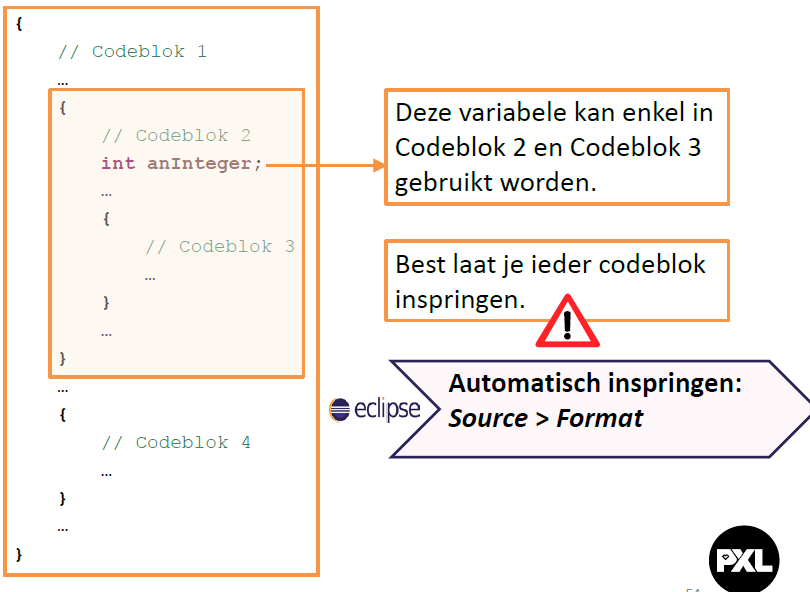
**5.4 Uitdrukkingen, statements en blokken**

In Java onderscheiden we uitdrukkingen, statements en blokken



Een codeblok is eigenlijk een groepering van een aantal statements tussen accolades. het heeft invloed op het bereik (scope) van een variabele (= gebied waar men gebruik kan maken van een variabele).

Voorbeeld:



**5.5 Programmaverloop statements**

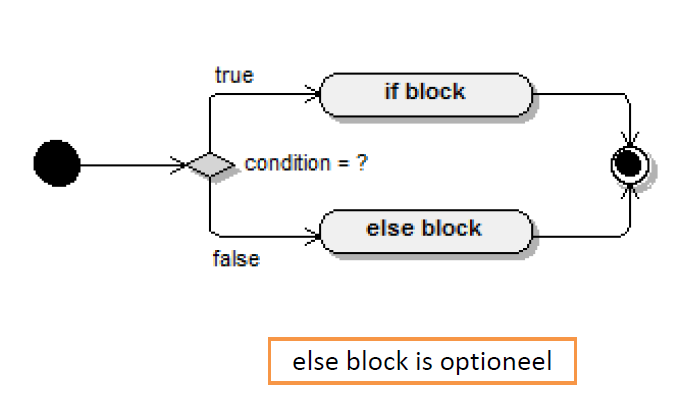
**5.5.1 Inleiding**

* if .. else (selectie)
* switch (selectie)
* while en do .. while (herhaling)
* for (herhaling)

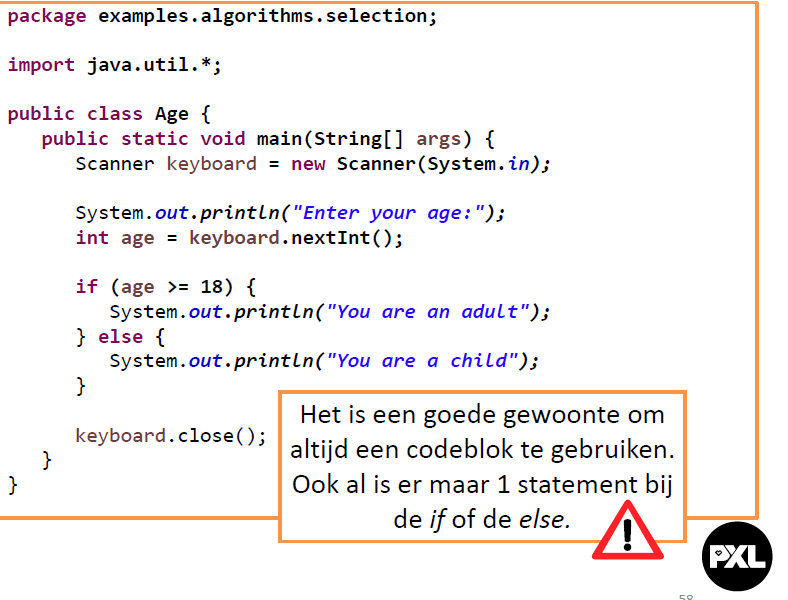
**5.5.2 Het if … else statement**

We kunnen een keuze maken in het uitvoeringspad met behulp van het if … else statement. Dit maakt het mogelijk om een bepaald statement of codeblok voorwaardelijk uit te voeren.

De voorwaarde is een boolean of een uitdrukking met als resultaat een boolean. Als de voorwaarde true is, wordt het statement of het codeblok uitgevoerd. Als dat niet zo is, en er is een else blok, zal dat uitgevoerd worden.



Voorbeeld met tip:

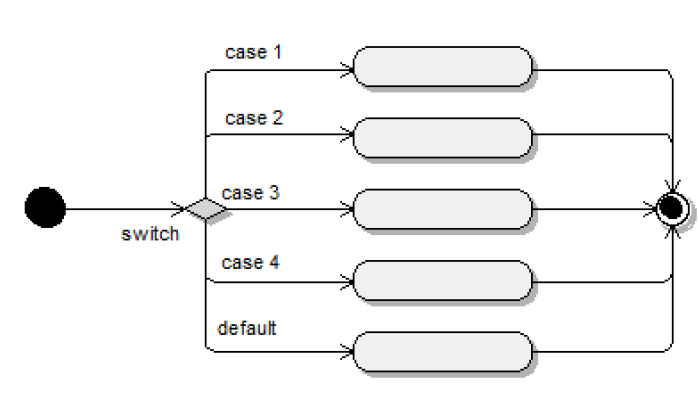


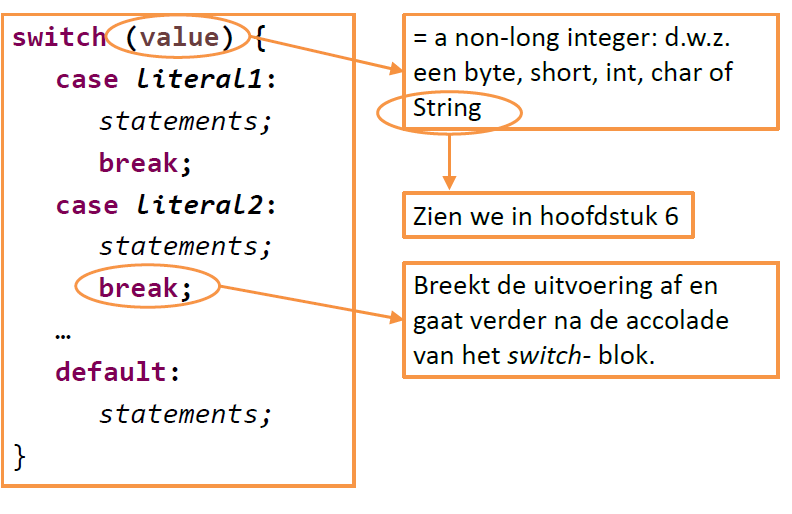
We kunnen if … else statements ook nesten

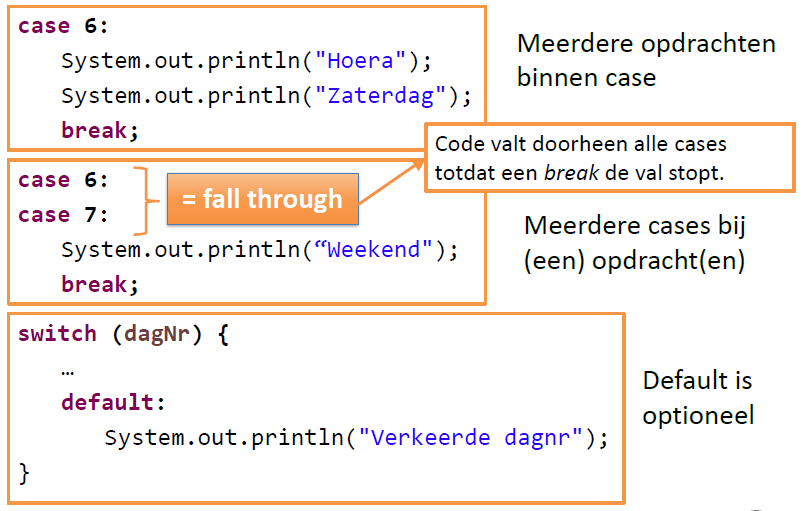


**5.5.3 Het switch statement**

Indien meervoudige keuzes gemaakt dienen te worden op basis van een integer-waarde of tekenreeks (string), dan kan dat gebeuren met een switch.

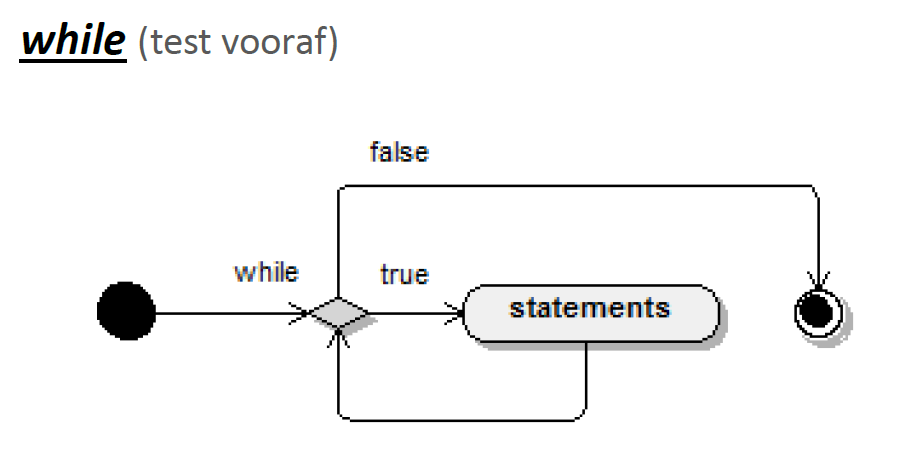






**5.5.4 Het while en do while statement**

Het while statement voert een statement of codeblok uit zolang een bepaalde voorwaarde waar is. We noemen dit ook wel een while lus.

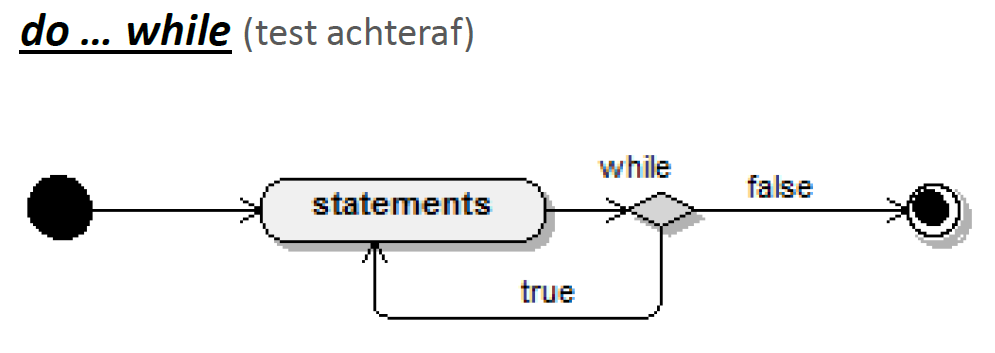


De voorwaarde is een boolean of een uitdrukking die als resultaat een boolean geeft. Als de voorwaarde true is, wordt het statement uitgevoerd. Nadien wordt de voorwaarde opnieuw geëvalueerd en als ze nog steeds true is, wordt het statement opnieuw uitgevoerd.



Een variant van het while statement is het do … while statement.

Bij het while statement wordt de voorwaarde geëvalueerd voordat het statement wordt uitgevoerd. Bij het do while statement wordt het statement eerst uitgevoerd en nadien wordt getest of het statement nogmaals uitgevoerd moet worden.

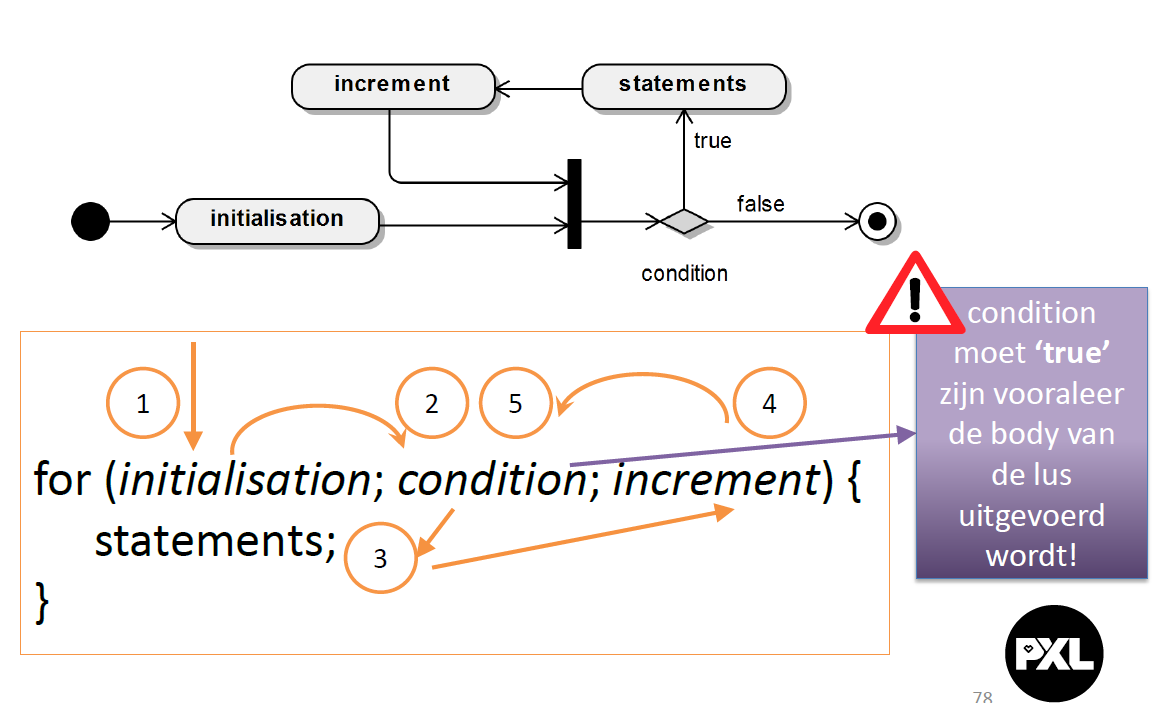


Het grote verschil met het while statement is dat het statement in dit geval minimaal 1x wordt uitgevoerd, ongeacht het resultaat van de voorwaarde.



**5.5.5 Het for statement of zelftellende lus**

Door een for statement, ook wel for lus genoemd, kunnen we op een eenvoudige manier een teller definiëren die automatisch verhoogd (of verlaagd) wordt bij elke herhaling.

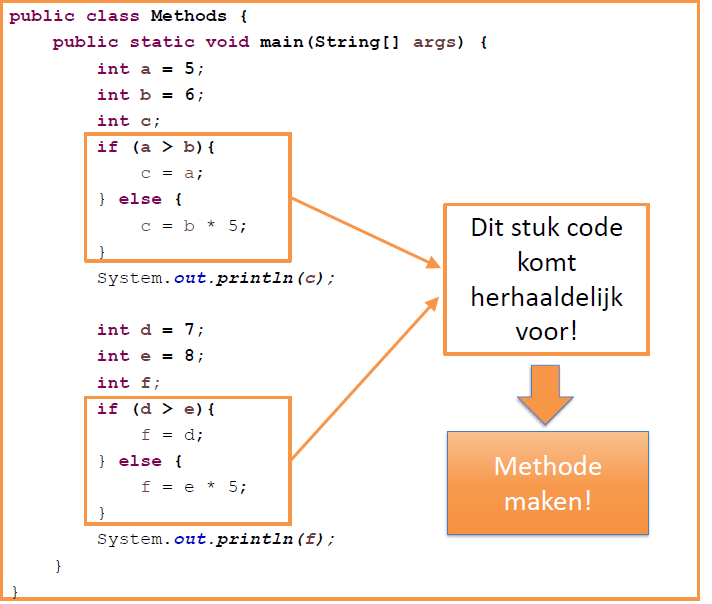


**5.6 Methoden**

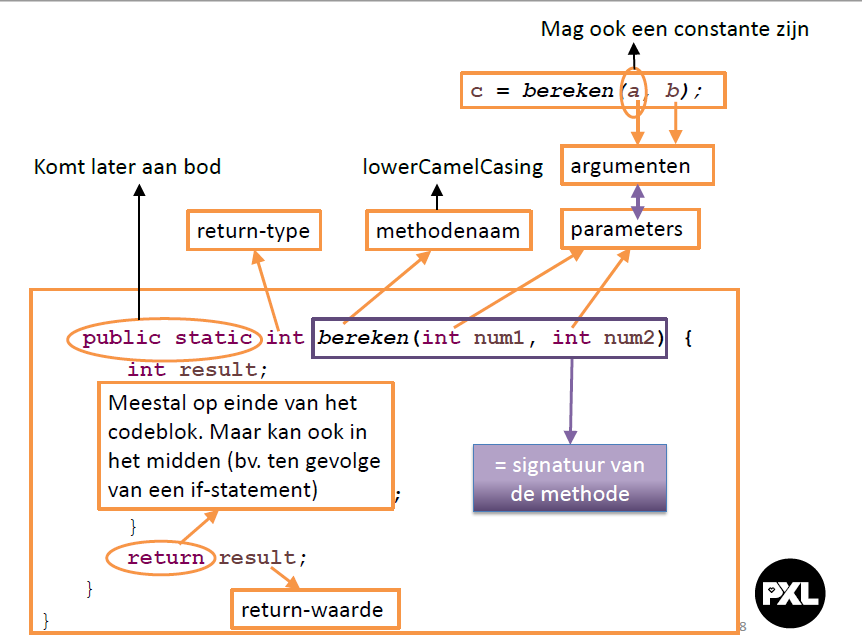
Vaak hebben we een situatie waarbij een reeks van programmeerregels telkens terugkomt.

We kunnen deze programmeerregels groeperen en deze groep een naam geven = methode.

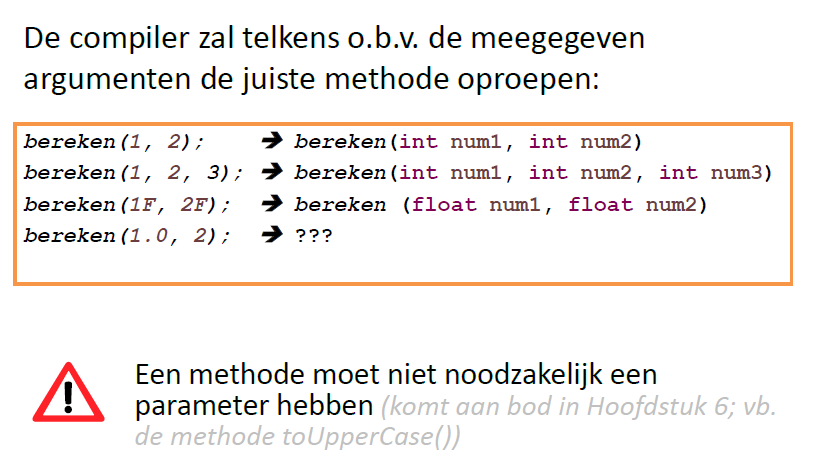
Een ander nut van methoden is dat ze ons toelaten om onze code te structureren. (een methode moet niet noodzakelijk herhaaldelijk uitgevoerd worden.).



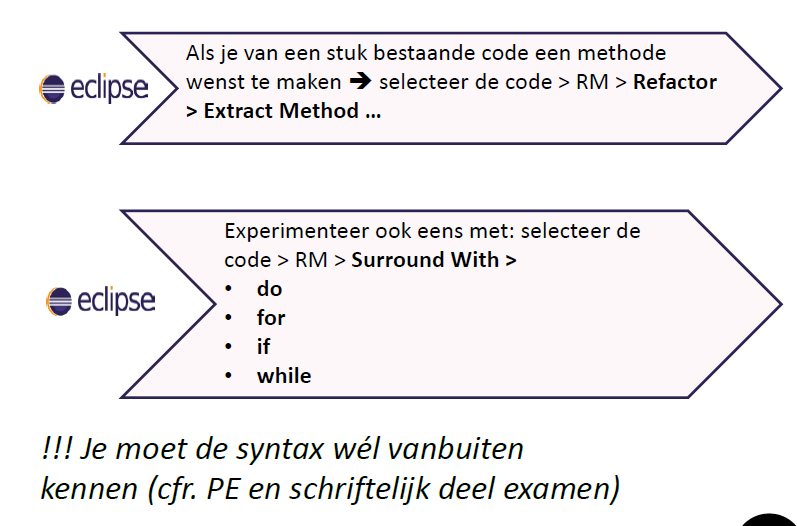












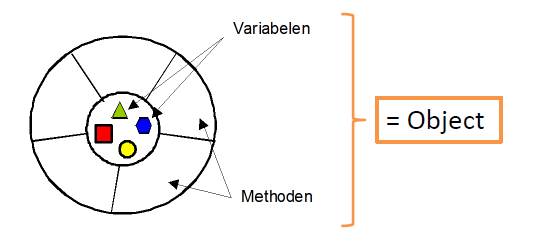
**Hoofdstuk 6: Objectgeoriënteerd programmeren**

**6.1 Inleiding**

* Inleiding in het objectgeoriënteerd programmeren
* Werken met bestaande objecten
* tekenreeksen

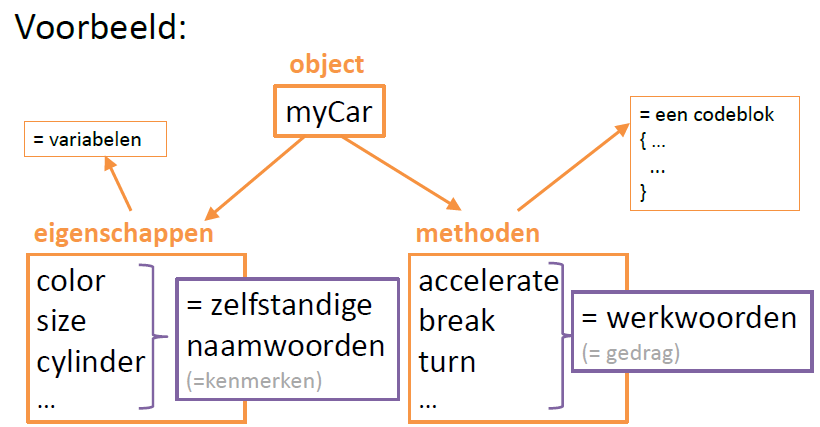
**6.2 Inleiding in het objectgeoriënteerd programmeren**

Het objectgeoriënteerd programmeren gebruikt het object als basis. Software-objecten kan men vergelijken met reële objecten of voorwerpen uit ons dagelijks leven. Zo is een auto een voorwerp.

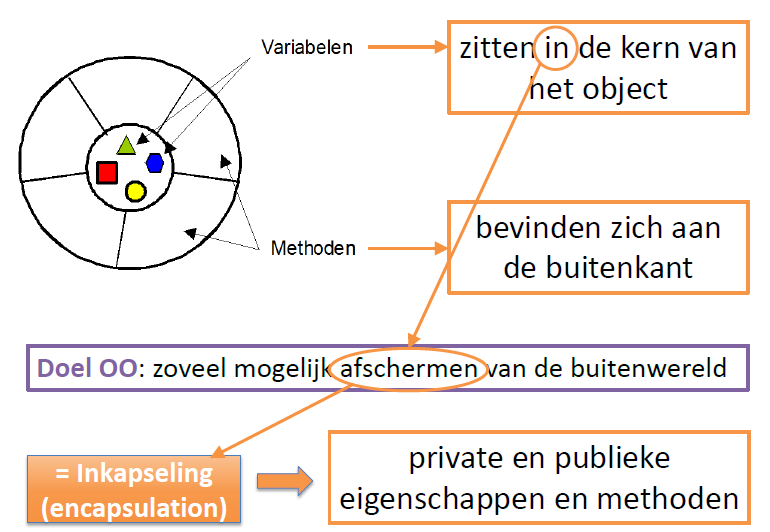


Een object is een groepering van variabelen (eigenschappen) en gerelateerde codeblokken (methoden)

lees uitleg p89



6.2.1.1 Private en publieke eigenschappen & methoden





**6.2.1.3 Voordelen van OOP**

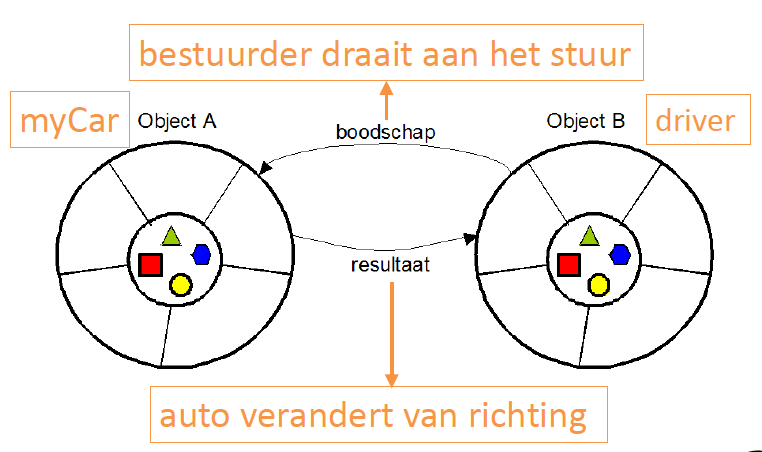
het objectgeoriënteerd programmeren heeft een aantal voordelen ten opzichte van het procedureel programmeren

* modulariteit
  + Code opdelen in modules, zodat deze herbruikbaar zijn.
* afscherming van informatie

**6.2.2 Boodschappen**

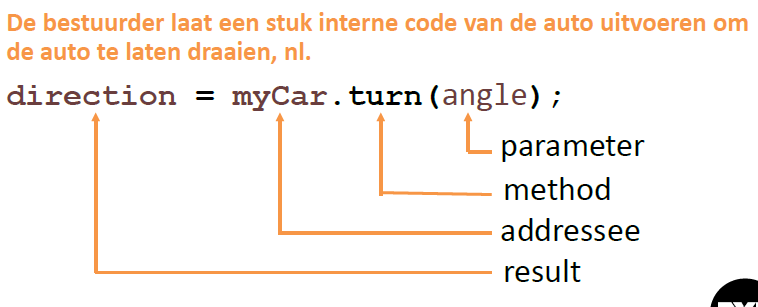
Een programma bestaat meestal uit verschillende objecten die met elkaar samenwerken en communiceren. Deze communicatie verloopt via boodschappen. Het ene object stuurt een boodschap naar het andere object om dat object iets te laten doen. Na afloop kan het ontvangende object het resultaat van de actie terugsturen.

Het zenden en ontvangen van boodschappen verloopt via het aanroepen van methoden van een object. Dit zijn blokken code die verbonden zijn aan het object.



Boodschappen hebben 4 kenmerken:

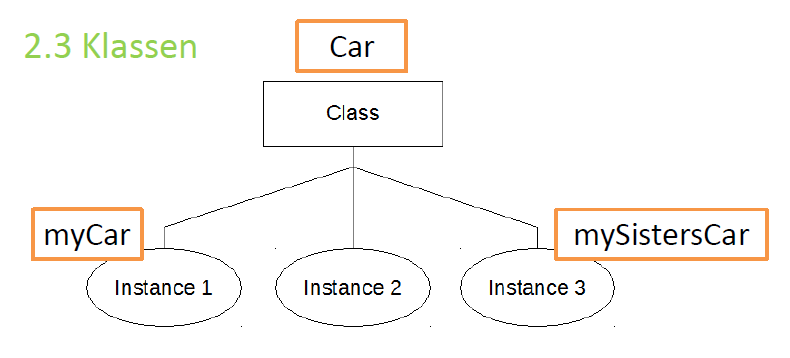
* De bestemmeling: het object waarde boodschap op gericht is
* De methode-aanroep
* bijkomende parameters die extra informatie geven over de boodschap (optioneel)
* het resultaat van de boodschap (optioneel)



**6.2.3 Klassen**

Veel voorwerpen uit ons dagelijkse leven zijn van dezelfde soort. Alle wagen zijn gefabriceerd met hetzelfde grondplan, op basis van dezelfde blauwdruk.

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



Door het onderscheid tussen klassen en objecten maken we ook een onderscheid tussen het soort variabelen:

* instantievariabelen (eigenschappen die kunnen verschillen per object)
  1. Bij een auto bv kleur, elke auto kan een andere kleur hebben.
* klassevariabelen (eigenschappen die gemeenschappelijk zijn voor alle auto’s van een bepaalde klasse)
  1. Bij een auto bv aantalWielen, elke auto van een bepaalde klasse heeft 4 wielen

En het soort methoden:

* instance methods (zijn gekoppeld aan een concreet object)
* class methods (hebben te maken met de klasse op zich)

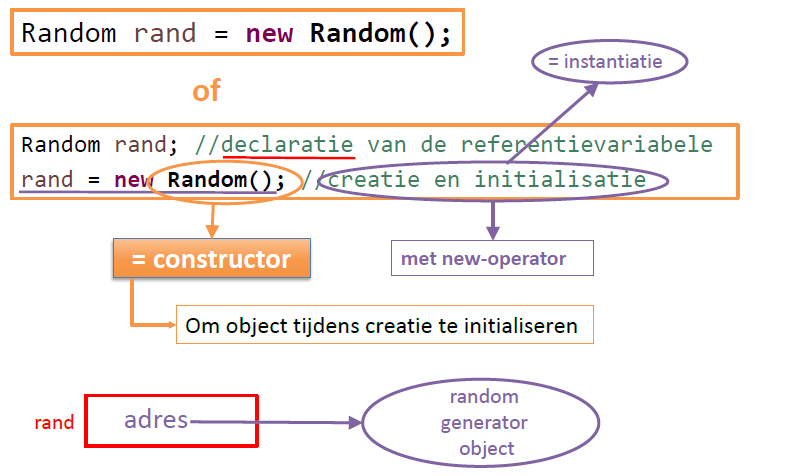
**6.3 Werken met bestaande objecten**

Objecten worden gemaakt op basis van een blauwdruk. Deze blauwdruk is de klasse waartoe het object behoort.

We maken een object van de bestaande klasse Random (bevindt zich in java.util). De volledige naam van de klasse is java.util.Random, de bytecode bevindt zich in Random.class

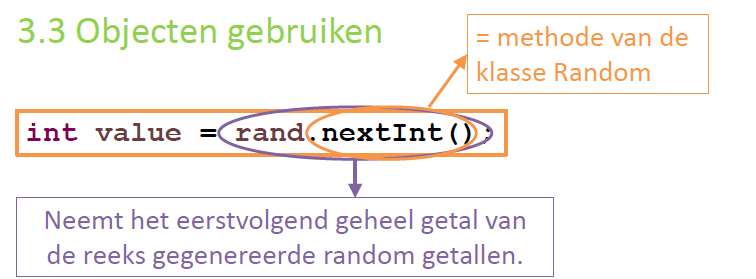
  

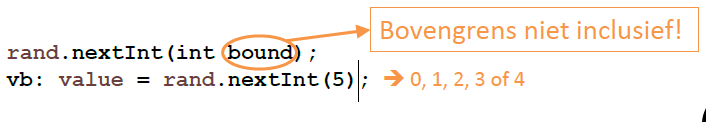

Tijdens de creatie wordt er geheugen gereserveerd voor het object. Het resultaat van de creatie is een verwijzing (referentie) naar het object in het geheugen.



**6.3.3 Objecten gebruiken**

We kunnen nu gebruik maken van de publieken eigenschappen van ons object.





Meer info in de javadoc

**6.3.4 Objecten opruimen**

Java heeft zijn eigen opruimdienst die het geheugen van objecten weer vrijgeeft als we het object niet meer nodig hebben. Als we een object niet meer willen gebruiken laten we ons object wijzen naar *null*.

**6.4 Tekenreeksen**

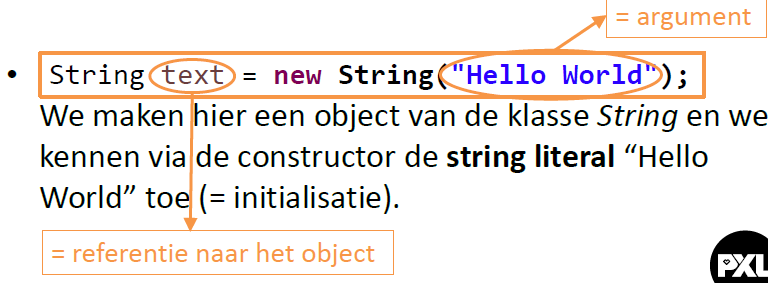
We behandelen 2 bestaande klassen voor het bewerken en opslaan van tekst

* String
* StringBuilder

**6.4.2 De klasse String**

* De String klasse wordt gebruikt voor onveranderlijke tekenreeksen. (immutable)
* Deze klasse behoort tot het pakket java.lang (standaard geïmporteerd)
* Bij de creatie van het object wordt een initiële waarde aan de string toegekend die nadien niet meer veranderd kan worden.

declaratie:



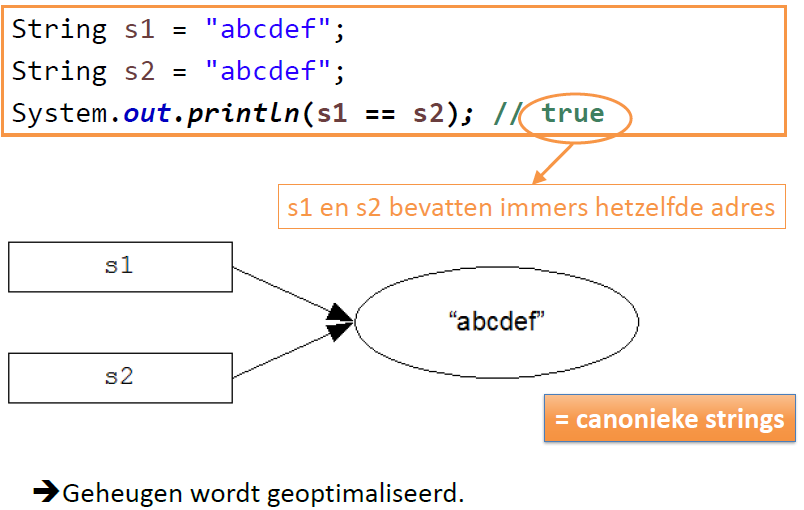


Aan de hand van JavaDoc moeten we in staat moeten zijn om alle methodes van de String klasse te kunnen gebruiken.

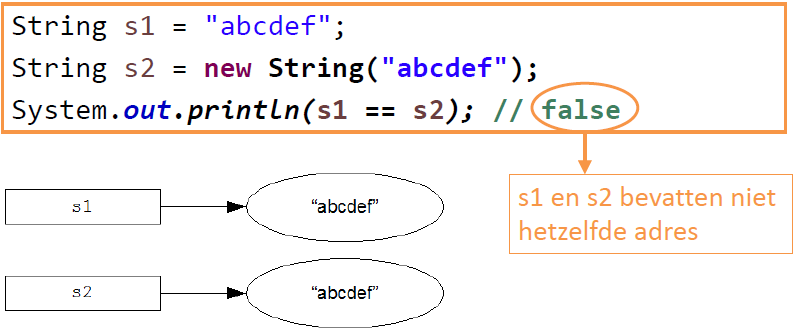
**!Het begin van een string heeft positie 0!**

**6.4.2.4 Geheugengebruik bij Strings**

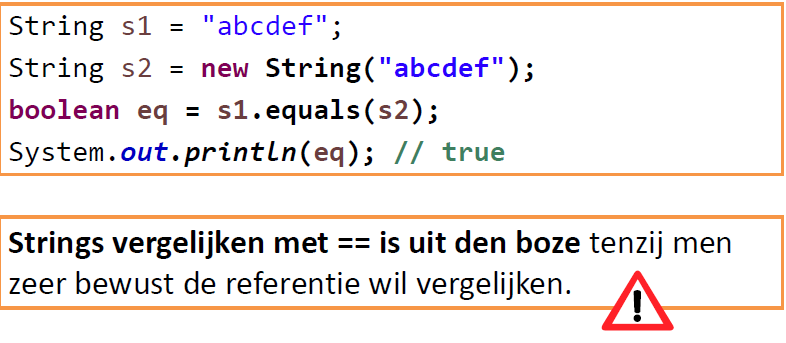
Omdat string immutable zijn, houdt de VM een pool bij van string-objecten : string constant pool. Zodra een nieuw String object wordt aangemaakt door middel van een string literal, gaat de VM na of deze string zich reeds in de pool bevindt en wordt er eventueel een referentie naar de reeds bestaande string teruggegeven.



Als men een String maakt door middel van de constructor wordt er wel een nieuw object gecreërd.



Als we de inhoud van twee Strings willen vergelijken, dan moeten we dit doen met de methode *equals*. Deze methode gaat namelijk de inhoud van de twee strings bekijken. Als we == gebruiken, vergelijken we de geheugenplaatsen en kunnen we alleen zien of twee objecten hetzelfde zijn of niet.

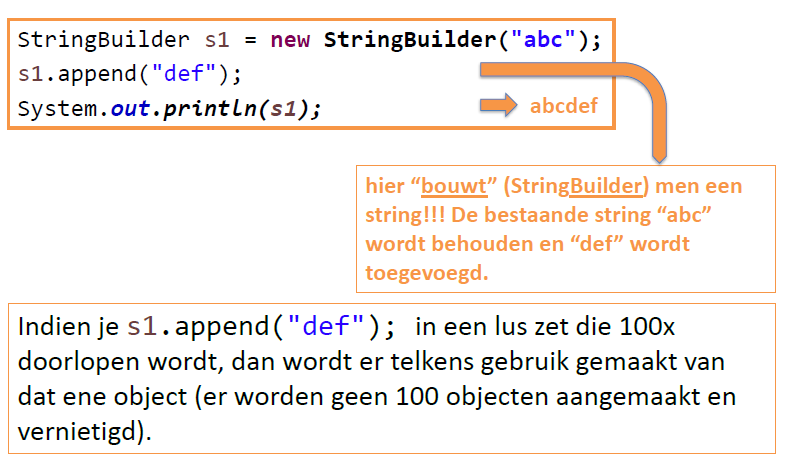


**6.4.3 De Klasse StringBuilder**

Objecten van deze klasse bevatten intern een tekenreeks die veranderd kan worden. De StringBuilder klasse heeft een aantal methoden die het mogelijk maken de string te wijzigen, zie JavaDoc.

StringBuilder vs String

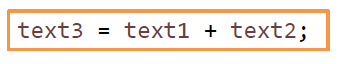




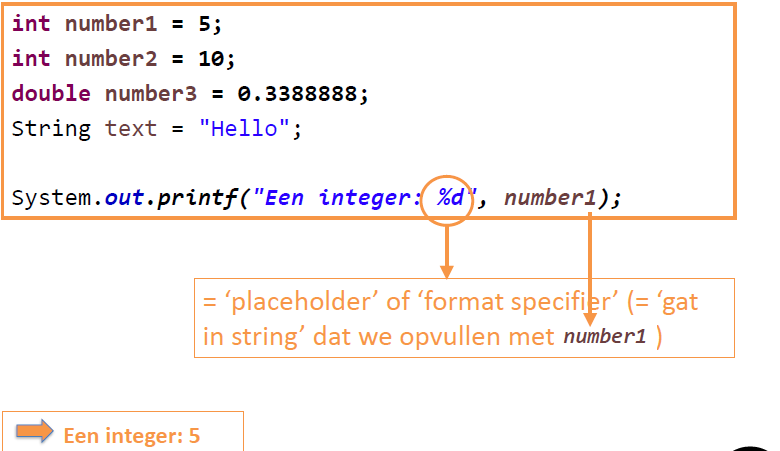


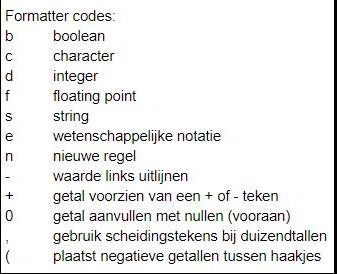
**6.4.4 Strings samenvoegen met de + operator**

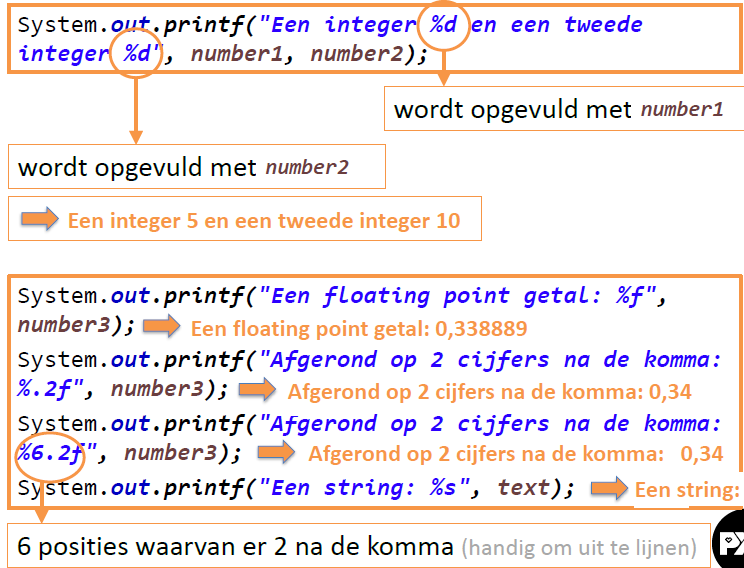
Stringbuilders kunnen worden samengevoegd met de append-methode. Het is ook mogelijk strings samen te voegen met de + operator.

****

**6.4.5 Gegevens formatteren met methode printf**





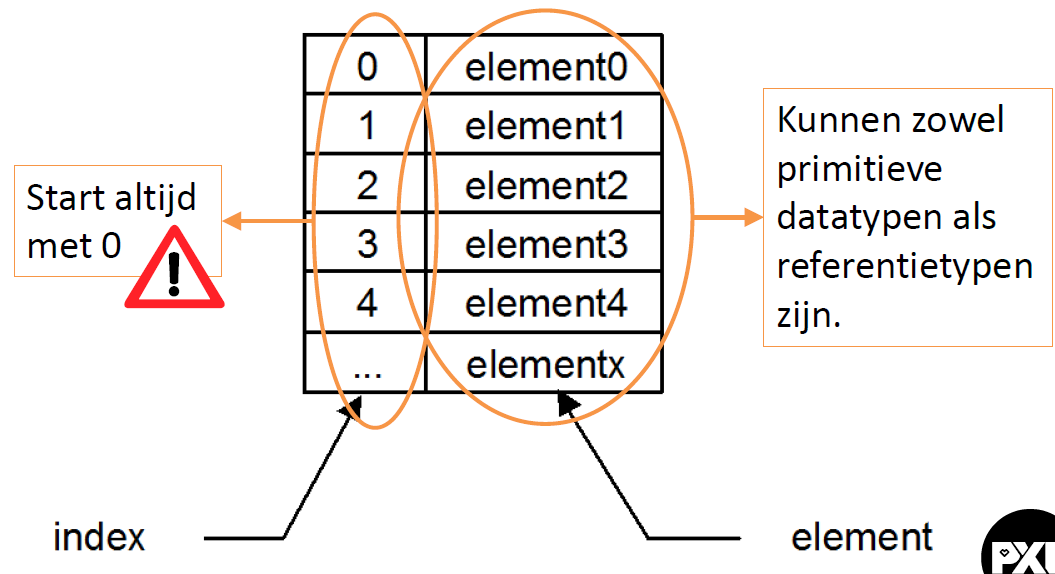




**Hoofdstuk 7: Arrays**

**7.1 Inleiding**

Een array is een verzameling van elementen van hetzelfde type waarbij ieder element voorzien is van een nummer (index) dat de plaats van dat element in de array aanduidt.

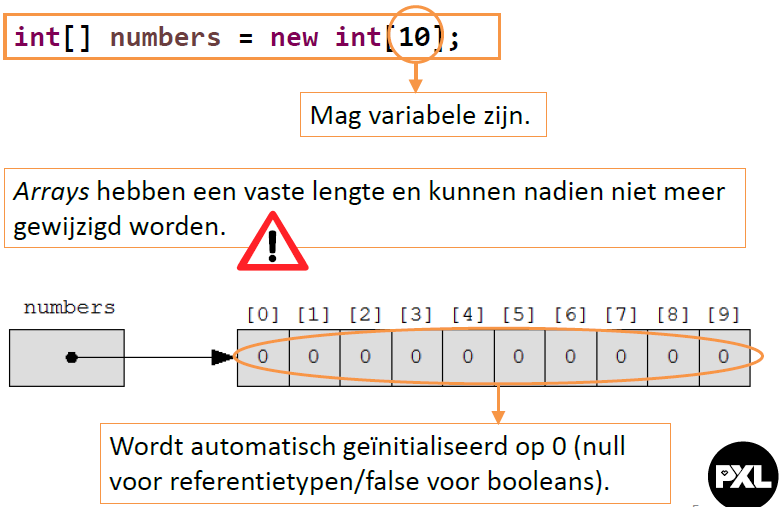


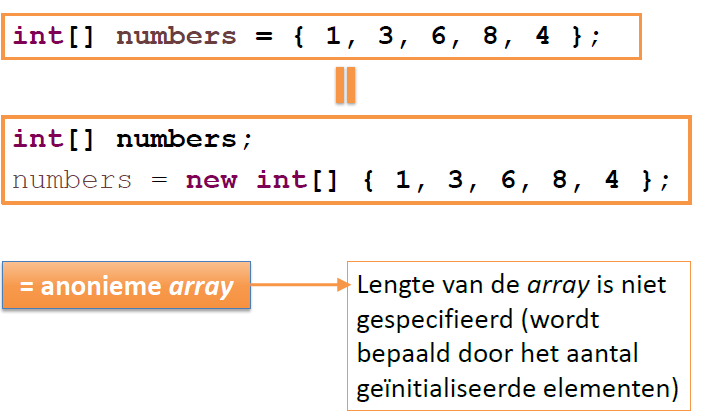
De elementen in een array kunnen zowel primitieve datatypen als referentietypen zijn. In het laatste geval heeft men een reeks van verwijzingen naar objecten.

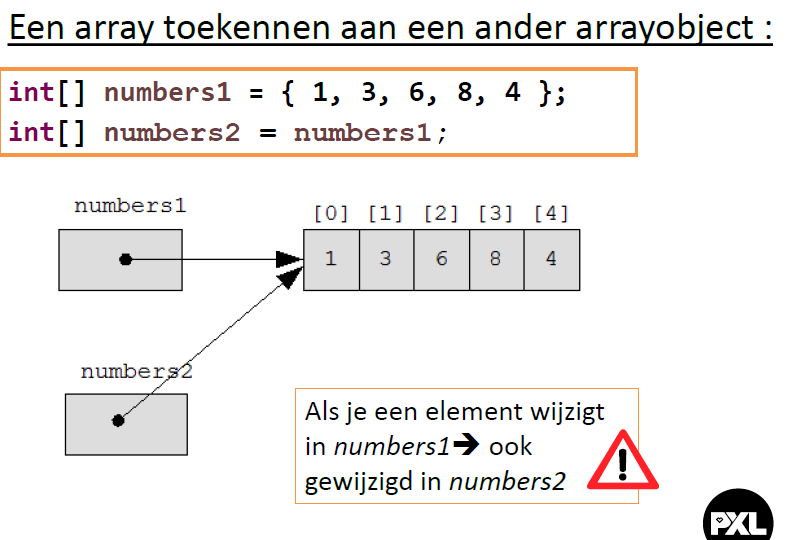
We gebruiken een array wanneer we gegevens in het geheugen moeten bewaren. Het zou onmogelijk zijn om dit te doen met individuele variabelen.

**7.2 Arrays maken**

Een array is op zich ook een object, arrays worden net als alle andere objecten gecreërd met de new -operator.



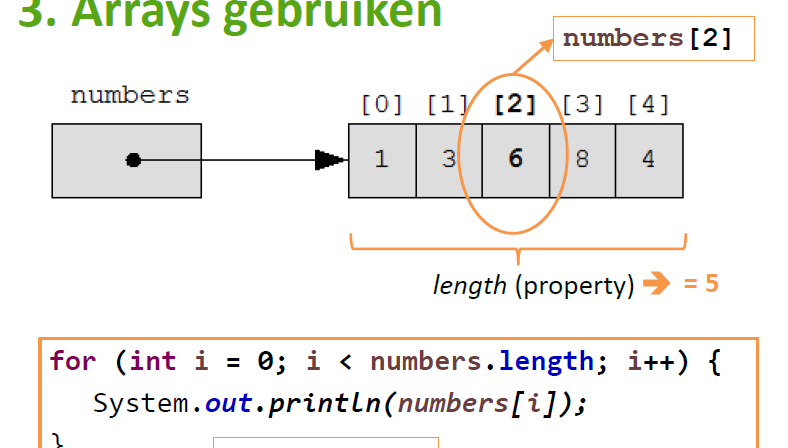
****



**7.3 Arrays gebruiken**

We kunnen de elementen van een array gebruiken door de naam van de array (referentie) te nemen, gevolgd door de vierkante haken met daartussen de index van het element dat we willen gebruiken: arrayName[index]

Een index van een element van een array begint steeds met 0.

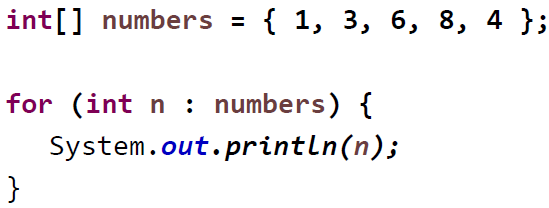


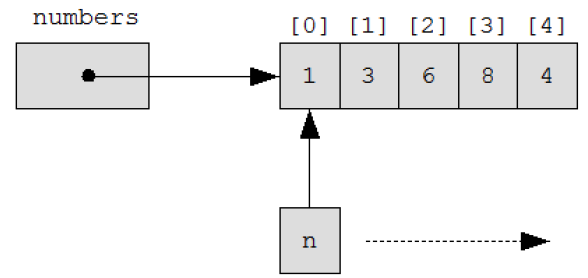
Alle arrays hebben een eigenschap die het aantal elementen van de array aangeeft: length

**7.4 De uitgebreide for - lus (for each)**

Om makkelijk over een array te loopen, kan men gebruik maken van de uitgebreide for-lus, ofwel for each-lus.

De variabele n neemt tijdens de iteratie één voor één de waarde aan van de elementen uit de array.



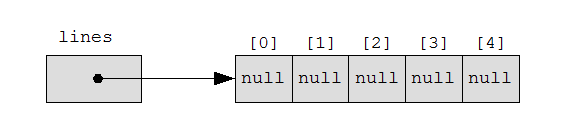


**7.5 Arrays van objecten**

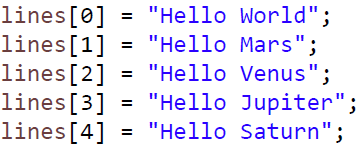
Een array van objecten bevat een aantal verwijzingen naar objecten van hetzelfde type.

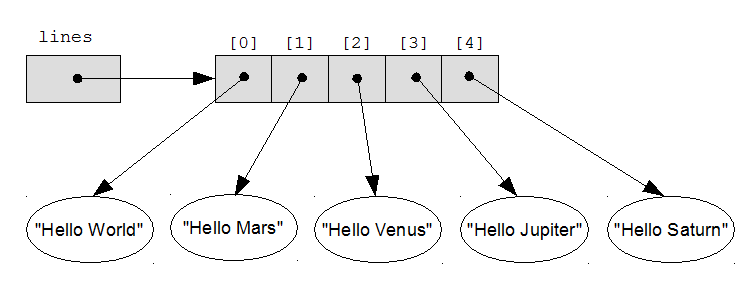
In het voorbeeld worden geen 5 String-objecten gecreëerd, maar een array van 5 referenties naar een String-object. Deze referenties worden geïnitialiseerd met de waarde *null*.



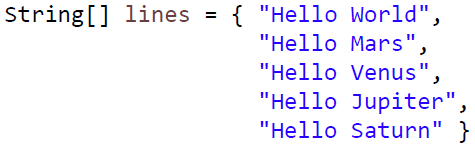


We moeten daarom vijf String-objecten afzonderlijk creëren en de referenties toekennen aan de elementen van de array.





Of we kunnen een array initialiseren tijdens de declaratie van de array

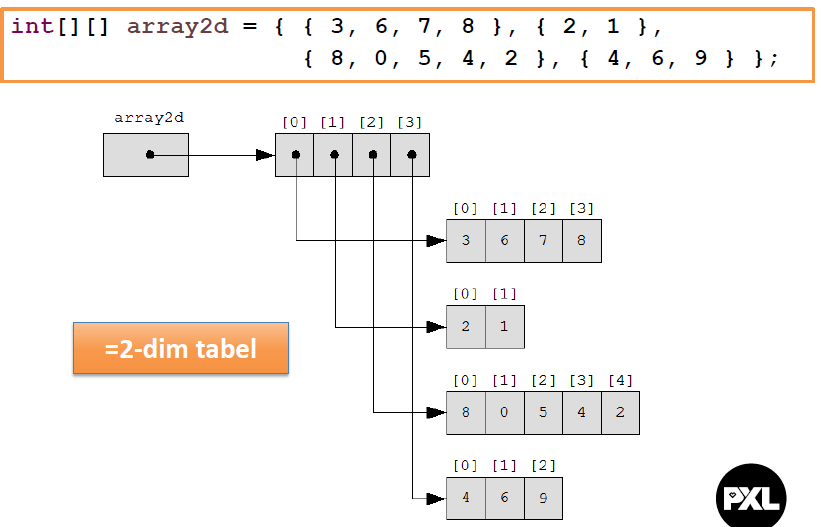


Voor meer methoden -> JavaDoc

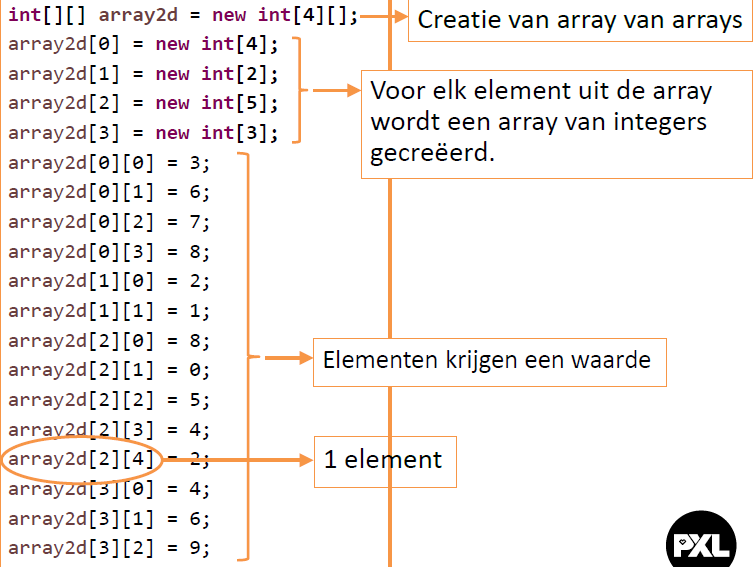


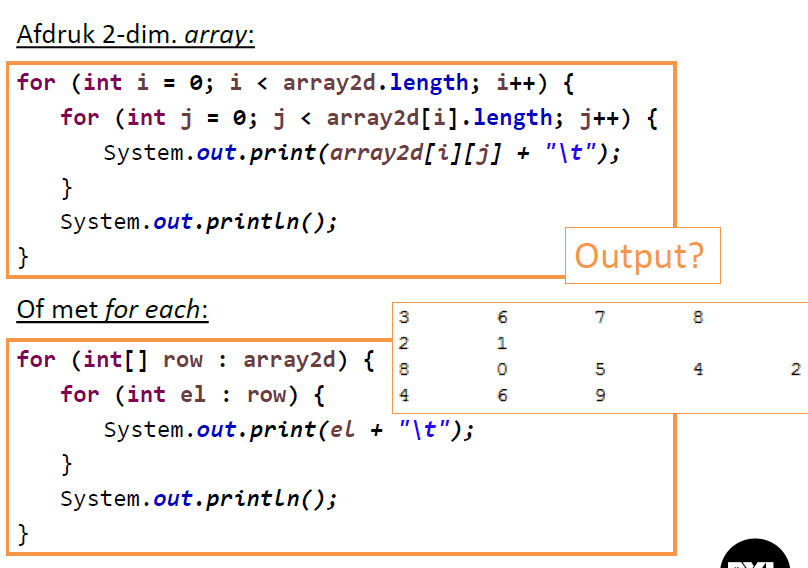
**7.6 Array van Arrays (meerdimensionale arrays)**

Vermits een array ook een object is, kan men ook een array van arrays maken.

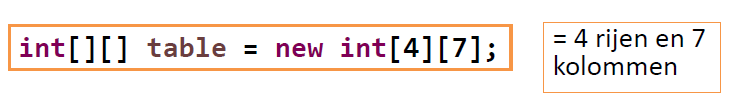


of



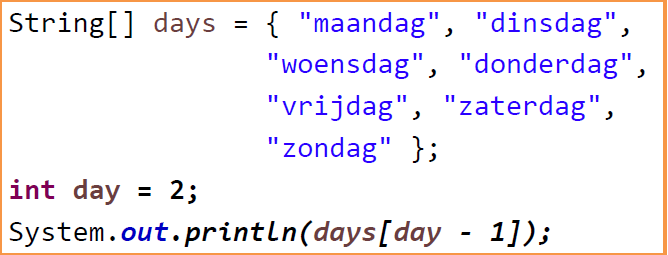


Indien de arrays allemaal dezelfde lengte hebben, kan men een tweedimensionale array als volgt declareren:



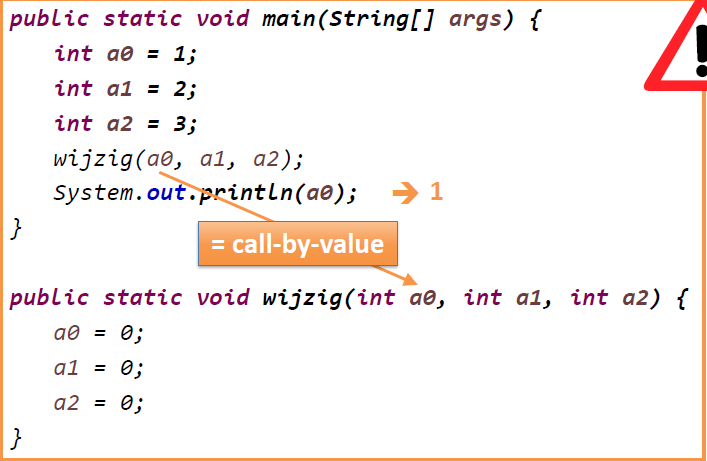
**7.7 Lookup tables**

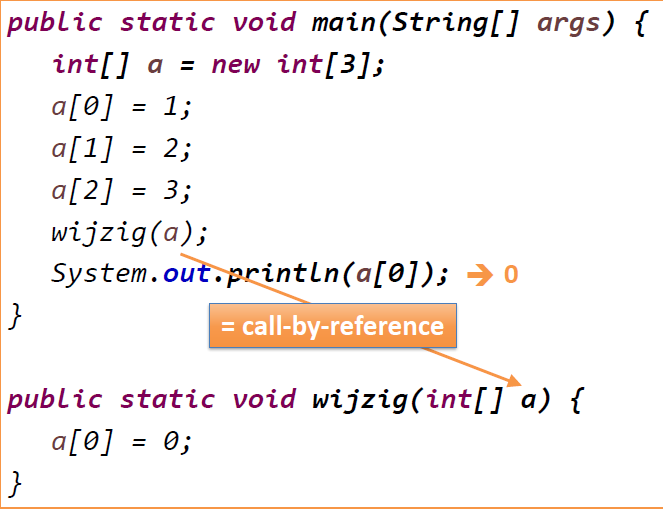
Arrays worden vaak gebruikt om gegevens op te zoeken en daarmee **kunnen lange if - else of switch case-constructies vermeden worden**. We noemen dergelijke arrays lookup tables. Gegevens kunnen dan eenvoudig **opgezocht worden aan de hand van hun index** in de array.





**!BELANGRIJK!**

****

****