H2

Comments	Je doet comments in een programma zodat ze beter leesbaar zijn Op 1 regel [end-of-line-comment] gebruik je // Voor meerdere regels gebruik je /* ….. */ , alles wat daartussen staat wordt genegeerd door de compiler
Class declaration	class – naam van de klasse. Alles in java is case sensitive
Method	void geeft weer dat de methode geen informatie teruggeeft, elke applicatie moet een main method hebben
Escape karakter	de \ backslash is een escape character. Het geeft aan bij println dat er een special character als output komt (bijvoorbeeld \n = newline)
printf	f betekent formatted, printf geeft aan dat er meerdere soorten dingen in kunnen staan, een %s staat voor string en %d voor een digit(getal)
import	een import declaratie geeft weer dat er een andere class o.i.d. gebruikt wordt in deze class
int, float, double	int = hele getallen, float = 32 bit met getallen achter de komma, double = 64 bit met getallen achter de komma
arithmetic operator	+, -, *, / en % . respectievelijk: Optellen aftrekken vermenigvuldigen delen en de remainder zoeken
relational operator	==, != , >, < , >=, <=. Respectievelijk: gelijk aan, niet gelijk aan, groter dan, kleiner dan, groter dan of gelijk aan, kleiner dan of gelijk aan

H3

Method	Als je een programma een taak wil laten uitvoeren heb je een methode nodig. in de methode kan je dingen stoppen die ervoor zorgen dat hij zijn taak uitvoert. Een methode kan een taak uitvoeren, kan een resultaat teruggeven, en kan parameters specificeren die de methode nodig heeft om zijn taak uit te voeren.
Class	Als je een klasse opslaat moet hij dezelfde naam hebben als hoe je die hebt genoemd in eclipse
%f	dit wordt gebruikt om een float of double uit te printen bij printf
acces modifier	public/private
class instance creation expression	Als je een object maakt van een klasse
dot seperator	wordt gebruikt bij een object om van dat object een methode aan te roepen

H4+5 door René, heb het blaadje overgetypt en alles een beetje uitgelegd. Een paar dingetjes snapte ik zelf ook niet helemaal dus daar heb ik de pagina van het boek bij gegeven zodat je zelf ook even kunt kijken.

4.3 **pseudocode**
(Pseudocode is een informele taal die je helpt met het ontwikkelen van algorithmes zonder dat je je zorgen hoeft te maken over de strenge gedetailleerdheid van de java syntax. Wat hiermee bedoeld wordt is dus dat je van tevoren precies gaat bedenken wat je programma moet doen (in normaal nederlands/engels), bijvoorbeeld: 'als de waarde groter is dan 1 dan moet het programma gaan runnen'. Deze taal werkt natuurlijk niet op de pc maar helpt je wel om te bedenken hoe je programma eruit moet zien zodat je het daarna makkelijker in java kan programmeren.)
4.4 control structures: **sequence, selection, repetition**
- sequential execution, transfer of control
(Normaal worden statements in een programma een voor een uitgevoerd in de volgorde waarin ze geschreven zijn, dit is sequential execution. Wanneer een programma gespecifieerd is dat het volgende uitgevoerde commando, niet de volgende in 'de rij' is, spreken we van transfer of control)
- **selection statements**: single, double, multiple
(Een selectie statement kan je gebruiken als je aan de hand van een gestelde conditie een keuze wilt maken uit een aantal alternatieven. Dit kan een if (single statement), if..else (double statement) of een switch (mutiple-selection statement) zijn.)
- **looping statements, loop-continuation condition**
(Java kent 3 soorten repetition statements (ook wel looping-statements genoemd), dit zijn statements die worden herhaald zolang er aan een bepaalde conditie (ook wel de loop-continuation condition genoemd) voldaan wordt. Deze 3 statements zijn de while, do..while en for statements)
4.6 IF..ELSE double-selection statement
- conditional expression: **ternary operator** (?:), three operands
(Java kent een conditionale expressie (?:) die gebruikt kan worden in een if..else statement. Dit is java's enige ternary operator (een operator die 3 operands nodig heeft), gezamelijk vormen ze een conditionele expressie: (studentgrade >= 60 ? "passed" : "failed");)
- **nested** if..else **statements**
(Een programma kan meerdere 'cases' testen door middel van een if..else statement te plaatsen in een andere if..else statement. Wanneer dit gebeurd spreken we van een nested if..else statement. Een voorbeeld (geschreven in pseudocode, waarvan we nu allemaal de betekenis weten) is:

```
If cijfer is hoger dan 9
        Print Teringjantjedikkenerd
Else
        If cijfer is hoger dan 8
        Print lolwutdunnenerd
        Else
        If cijfer is hoger dan 7
                Print mwahgewoonslim
        Else
                Etccccccccc)
```

- **dangling else problem**
(Wanneer java een else ziet, zoekt hij onmiddelijk naar een if-statement, dit kan worden voorkomen door het plaatsen van brackets ({ }), dit kan leiden tot en dangling-else problem,

kijk voor een voorbeeld even op pagina 114 van het boek, want ik zou niet zo goed weten hoe ik dit kort kan opschrijven)
- **syntax error, fatal/nonfatal logic error**
(Syntax errors, bijvoorbeeld een bracket is weggelaten) worden ontdekt door de java compiler. Een logic error (wanneer bijvoorbeeld 2 braces in een block weggelaten zijn) hebben effect wanneer het programma wordt uitgevoerd. Een fatal logic error zorgt ervoor dat het programma faalt en onmiddelijk termineert. Een nonfatal logic error zorgt ervoor dat het programma door kan gaan, maar geeft foute resultaten.)
4.7 WHILE repetition statement
Een repetition statment is een soort loop, die elke keer controleert of de conditie waar is, is dit het geval zal hij doorgaan met loopen, wanneer de conditie fout gezet wordt, stopt hij.)
4.8 counter-controlled repetition
- **integer division**, truncation
(wanneer 2 integers gedeeld worden spreken we integer division, voorbeeld op pagina 122 van het boek)
4.9 sentinel controlled repetition
- indefinite repetition, **sentinel** value
(Wanneer een gebruiker gevraagd wordt gegevens in te voeren (bijvoorbeeld een cijferlijst) moet er een speciaal command worden toegevoegd om java te laten weten wanneer we klaar zijn met het invoeren van de cijfers. Dit kunnen we doen d.m.v. een sentinel waarde, een waarde dat de gebruiker invoert zodat de computer weet dat we klaar zijn met het invoeren van cijfers.)
- pseudocode: top-down **step-wise refinement**
- explicit conversion: **unary cast operator** (...)
(Wanneer er een gemiddelde uitgerekend moet worden zal je het probleem krijgen dat java met standaard integers niet met getallen achter de komma kan werken. Hiervoor biedt java een unary cast oparator, een soort tijdelijk floating-point dat het getal rechts van de komma onthoudt)
- implicit conversion: **promotion**
(java kan alleen rekenen met artithmetische expressies, waarvan de operands types hetzelfde zijn. Om zeker te zijn dat dit het geval is kent java een operation die promotion heet, die uigevoerd wordt op de geselecteerde operands) (als je hier geen kut van snapt, lees dan 4.8 t/m 4.10 door, leuker kunnen we het niet maken, makkelijker helaas ook niet)
4.12 increment and decrement operator
- **prefix, postfix, increment, decrement**
(Java kent 2 unary operators voor het 1 toevoegen of 1 aftrekken van een value van een nummeric variable. Dit zijn de increment operator (++) and the unary decrement operator (--). Je kunt hier dus gewoon c ++ typen in plaats van het wat lastigere c = c+1. Een increment of decrement operator dat voor een variable geplaatst is (prefixed) wordt ook een prefix increment of prefix decrement operator genoemd. Wanneer een increment of decrement variable achter een variable is geplaatst wordt dat ook wel een postfix increment of postfix decrement operator genoemd.)
4.13 primitive types
- **strongly types language**
(Een taal waarin alle variables een type moeten hebben)

5.2 essentials of **counter-controlled repetition**
(Naam zegt het al, iets dat constant herhaald wordt en stopt totdat de waarde van de gezette counter is bereikt, hiervoor is benodigd: een variable, ook wel counter of control variable genoemd, die bepaalt hoe vaak een set of statements wordt uitgevoerd. Een initial

value van deze variable. Een increment of decrement (die de control variable elke keer bewerkt als deze door de loop komt). En de loop-continuation condition die controleert of de loop moet doorgaan.)
5.3 FOR repetition statement
- **off-by-one error**
(Als je bijvoorbeeld de conditie van een loop <=10 opschrijft als <10, hierdoor zal de loop namelijk maar 9 keer runnen in plaats van 10 keer.)
- **scope of a variable**
(Als de initialisatie expressie in de for header een control variable declareert, kan deze control variable alleen gebruikt worden in deze for statement. Hij zal hierbuiten niet bestaan. Dit gelimiteerde gebruik wordt ook wel een variable's scope genoemd (pagina 161))
5.4 examples using FOR statements
- **formatting flags**: minus sign (-), comma (,)
(Ik kan hiet wel een onzin verhaal gaan neerplanten over field withs, maar ik snap deze zelf niet 100%, de uitleg staat onderaan pagina 165 van het boek, je kunt dus beter zelf even kijken)
5.5 DO..WHILE repetition statement
(Een statement dat herhaald wordt zolang er aan een bepaalde eis voldaan wordt, deze is anders dan een normale while statement: In een while statement controleert het programma aan het begin van het script of hij verder moet gaan met de loop, dus voordat de loop's body wordt uitgevoerd. Als de conditie van de loop-continuation statement false is, wordt de body dus nooit uitgevoerd. Bij de do...while statement gebeurd deze controle aan het eind van de loop's body. Dus na het uitvoeren van het programma. Hierdoor wordt de body dus altijd minstens een keer uigevoerd.)
5.6 SWITCH multiple selection statement
- **falling through**
- **utility, helper methods**

(deze kon ik niet vinden :S?)
5.7 BREAK and CONTINUE statements
(Een break statement zorgt voor een onmiddelijke exit van de statement. Een continue statement slaat de overgebleven statements in de loop body over en gaat verder met de volgende iteration van de loop.)
5.8 **logical operators**
(De logical operators in java zorgen dat je complexere condities kan vormen door meerdere normale operators te combineren. Bijvoorbeeld && = conditional AND, || = conditional OR, & = boolean logical AND, | = boolean logical inclusive OR, ^ = boolean logical exclusive OR en ! = logical NOT)
- conditional AND (&&), condidtional OR (||): short-circuit evaluation
- **side effect**
(Een modificatie van een variables waarde)

H6

Divide and Conquer	Ervaring heeft getoond dat de beste manier om een groot programma te ontwikkelen en onderhouden is om het te maken van kleine makkelijke stukjes, dit heet divide and conquer
Packages	klassen zijn vaak in packages gegroepeerd zodat ze geimporteerd en opnieuw gebruikt kunnen worden.

Static	Een klasse kan static methodes bevatten om veelvoorkomende taken te volbrengen die geen object van die klasse nodig hebben. Je kunt een static methode aanroepen door klasse.methode(argument) te typen.
Field	Alle variabelen zijn de fields van een klasse
Methode	Een methode kan max. 1 waarde teruggeven, maar die waarde kan refereren naar een object met vele waardes.
Field Variabele	Een variabele zou als field gedeclareerd moeten worden als hij in meer dan 1 methode gebruikt wordt.
Stacks	LIFO, Last in, First out. Insert = push, remove = pop from stack
Random	Objecten van de klasse random kunnen random int, long, float of double waardes produceren
Enumeration	Een enumeratie herken je aan het keyword enum en een type naam. Na de enum declaratie + de naam komen er { } haakjes voor de body van de enumeratie. In de haakjes is een lijst [gescheiden door komma's] met enumeratie constanen. Elke representeert een unieke waarde. Constanen kunnen als public final of static aangegeven worden. Deze zijn dan volledig in hoofdletters om ze te onderscheiden van de rest van t programma.
Scope	Scope is het deel van een programma waarin een eenheid zoals een variabele of een methode aangeroepen of doorverwezen kan worden bij zn naam. Zo'n eenheid is dan in scope van het programma. De scope van een parameter declaratie is de body van de methode waarin de declaratie voorkomt De scope van een local-variable declaratie is van t begin tot t eind van dat blok
Overloading	Java staat toe dat methodes dezelfde naam hebben (overloading), zolang ze verschillende soorten parameters hebben.

Samenvatting H7 Arrays and ArrayLists

7.1
Data structures = Verzameling van verbonden data, bijvoorbeeld Arrays.

7.2
Array = Groep variabelen (**elements|components**) die waarden bevatten van hetzelfde type.
Subscript|index = Positie nummer van element in een array.
Array-acces expression = Om te refereren aan een element in een array. Bevat naam vd array gevolgd door de **index** ingesloten door **square brackets** (**[ ]**). Eerste element in een array heeft altijd **index zero** en is **zeroth element**.

7.3
Array-creation expression = Wordt gebruikt bij het aanmaken van een array. Deze expression geeft een reference die wordt opgeslagen in een array variable. Bijv: (int[] c = new int[12];)

7.4

(fig. 7.2 t/m 7.8)
Array initializer = Hiermee kan je de elementen in een array initialiseren. Bijv: (int[] n = {10, 20, 30, 40, 50}; Het laatste deel tussen accolades is de **initializer list**. In dit geval bepaalt die de lengte van de array.
Named Constant = Constant variable. Maken programma´s beter leesbaar. ARRAY_LENGTH is bijvoorbeeld duidelijker dan literal value (bijv. 10).

7.5
(fig. 7.9 t/m 7.11)

7.6
(fig. 7.12)
Enhanced for statement = Loopt door de elements van een array zonder gebruik te maken van een counter. Hierdoor is " stepping outside" de array niet mogelijk. Een Enh. for statement kan alleen worden gebruikt om array elements te verkrijgen en niet om ze te modificeren. Dat is ook het verschil met een standaard for statement.

7.7
(fig. 7.13)
Scalar|scalar quantities = een copy van de element´s value van een primitive type.
Pass-by-value = Een copy van element´s value wordt gegeven aan de called method. Bij modificatie verandert alleen de copy en niet het origineel. P-B-V is gebruikelijk in Java. Dit is tijdrovend want alle data moet worden gedupliceerd.
Pass-by-reference = De called method heeft toegang tot de argument´s value. En deze kan worden veranderd ipv de copy. Dit is niet mogelijk in Java.

7.8
(fig. 7.14, 7.15)
7.9
Multidimensional arrays = Wordt vaak gebruikt om waarden in tabellen weer te geven. Bijvoorbeeld in een **two-dimensional array** waar twee indices worden meegegeven voor row en collum. **M-by-n arrays** hebben m rows en n arrays.

7.10
(fig. 7.18, 7.19)

7.11
(fig. 7.20)
Ellipsis = Een data type gevolgd door een ellipsis… (bijv. average (double… numbers) geeft aan dat de method een onbekend aantal arguments ontvangt van dat type. Hetzelfde is te bereiken door method overloading en array passing, maar werken met een ellips is beknopter.

7.12
(fig. 7.21)

Command-line arguments = Argumenten die worden gepassed vanaf de command line.

Samenvatting hoofdstuk 8

Een **this reference:** Elk object heeft toegang tot een verwijzing naar zichzelf met het *keyword* **this (**of **this reference.)** Als een *non-static* methode wordt aangeroepen voor een bepaald object, wordt impliciet gebruikgemaakt van het keyword **this** in de *body* van de methode om te verwijzen naar de instantievariabelen van dat object en andere methoden. Je kan **this** ook expliciet gebruiken in de *body* van een *non-static* methode. Het **this** keyword kan niet gebruikt worden in een *static* methode.
Shadowing instantie variabelen: Het verbergen van instantie variabelen met behulp van **this,** bijvoorbeeld wanneer de parameternamen in de constructor gelijk zijn aan de instantievariabelen uit de desbetreffende klasse. (zie pagina 323-324 voor een voorbeeld)
Overloaded constructors: Dezelfde naam voor een methode gebruiken door andere parameters, een ander aantal parameters of een andere volgorde van parameters te gebruiken. (zie pagina 325-326 voor een voorbeeld)
Default values: Als je geen constructor in een klasse neerzet, maakt de compiler er een voor je met standaard waarden (**default values).** Een int wordt bijvoorbeeld 0 en een boolean false.
Compositie (ookwel **has-a relationship):** Een klasse kan verwijzingen hebben naar objecten van andere klassen als *members.* (een voorbeeld staat op pagina 332)
Klassen en enum typen: Een enum type wordt gedeclared met een **enum declaration.** Dit een door komma's gescheiden lijst van enum-constanten. De volgende beperkingen zijn belangrijk bij de **enum declaration**:
- enum constanten zijn altijd impliciet **final**, omdat ze natuurlijk constanten maken die niet aangepast hoeven/mogen worden.
- enum constanten zijn altijd impliciet **static**
- Een error zal voorkomen wanneer een object wordt aangemaakt van een enum typen waarin *new* staat.

Static field: Normaal gesproken heeft elk object zijn eigen kopie van de instantievariabelen van de klasse waarvan het een object is. Soms wil je dit niet. In dat geval gebruik je een **static field (met een class variabele)**. Je declareert een **static variabele** met het sleutelwoord *static.*
Principle of least privilege: In een programma moet een stuk code zo min mogelijk privileges hebben, maar toch voldoende om zijn taak te voltooien. Hiermee wordt de code een stuk robuuster gemaakt en is de kans kleiner dat het stuk code variabelen gaat aanpassen die het niet hoort aan te passen. **Final** variabelen helpen bij het maken van robuuste code. (voorbeeld op pagina's 344-345)

Name conflict, name collision: Deze termen betekenen hetzelfde: als je meerdere packages gebruikt die bijvoorbeeld allebei de klasse Time1 bevatten, zal dit een **name conflict** opleveren als je Time1 aanroept. Dit kan voorkomen worden door de complete packagenaam met daarachter Time1 te gebruiken.

Modifiers private, public, protected en default access: Zie figuur 8.20 op pagina 351.

Samenvatting hoofdstuk 9

Inheritance: Een handige functie bij object georienteerd programmeren. Je kan een klasse (de **subclass)** eigenschappen van een andere klasse (**de superklasse**).

Class hierarchy: Beschrijft de overervings relaties tussen klasse.

Single-inheritance: Dit is het enige type overerving wat Java ondersteunt: je kan alleen van een directe **superklasse** overerven.

Is-a vs. Has-a relationship: Is-a representeert overerving: een object van een subklasse is ook een object van de superklasse. Voorbeeld: Een auto is een voertuig.
Een has-a relationship representeert *composition* (zie hoofdstuk 8): een object heeft referenties naar *members* van andere objecten. Voorbeeld: Een auto heeft een stuur.

Class libraries: Een 'bibliotheek' waarin klassen zijn opgeslagen waarvan dan weer gebruik gemaakt kan worden door programma's.

Inheritance hierarchy: Een boom waarin je ziet wat waaronder valt, zoals ook een beetje met XML. (voorbeeld op pagina 368)

Package access: Als je een impliciete modifier 'gebruikt' dan krijgt de package default package access. Als je expliciete modiers gebruikt, dan kies je uit de volgende: **private, protected** of **public access.** (zie pagina 369-370 voor meer uitleg)

Object: Alle klassen erven uiteindelijk over van de klasse **Object,** dit is de hoogste superklasse in Java**.** (java.lang)

Shallow copy vs deep copy: In de klasse Object zit de method *clone.* Deze kan een zogenaamde **shallow copy** en een **deep copy** maken.
Shallow copy: Dit is de default implementatie. De waarden van instantievariabelen worden in een ander object van hetzelfde type gekopieerd. Bij een referentietypen= wordt alleen de referentie zelf gekopeerd.
Deep copy: Een **deep copy** maakt een nieuw object aan voor elke referentie-type instantie variabele (zie 392 voor meer informatie).

H10

Polymorphisme	polymorphisme zorgt ervoor dat je meer algemeen kunt programmeren
Interface	Een interface beschrijft een set van methoden die aangeroepen kunnen worden op een object, maar die geen concrete implementatie voor die methoden geven. Je kan klasses declareren die 1 of meerdere interfaces implementeren.

Key concept	Een object van een subklasse kan gebruikt worden als een object van zijn superklasse, maar niet andersom! [is-a , not has-a]
Downcasting	Downcasting zorgt ervoor dat een programma methodes kan gebruiken die niet in de superclass zitten.
Abstract classes	Abstracte klasses zijn klasses die je declareert als je er geen objecten van wilt maken. Abstracte klasses zijn incompleet, de missende delen moeten de subklasses invullen, om concrete klasses te worden. Abstracte methoden worden overschreven. Deze moet je declareren in een subklasse, anders wordt deze subklasse ook abstract.
Dynamic binding	Ook wel late binding. Java kiest ervoor om op het 'laatste' moment te kiezen wat voor methode uit te voeren, tijdens execution time, in plaats van tijdens compilation time.
Final	final variabelen kunnen niet gewijzigd worden nadat ze geinitialiseerd zijn. Een methode die met final gedeclareerd is kan ook niet gewijzigd worden. De final method declaratie kan niet gewijzigd worden, dus alle subklassen gebruiken dezelfde method implementation, dit heet static binding Een klasse die met final is gedeclareerd kan geen superklasse zijn.
Interface declaration	Een interface declaratie begint met het woord interface, en kan alleen constante en abstracte methodes bevatten. Alle klassen moeten public zijn, en interfaces mogen geen details van implementatie geven. (zoals variabelen) Als je bij een klasse een interface gebruikt, moet je al die methodes gebruiken, anders wordt je klasse abstract.
Implementatie	Realisatie van een interface
Interface vs abstract	Interfaces worden vaak in plaats van abstracte klassen gebruikt, dit gebeurd vaak als er geen default methods of fields zijn in de klasse.