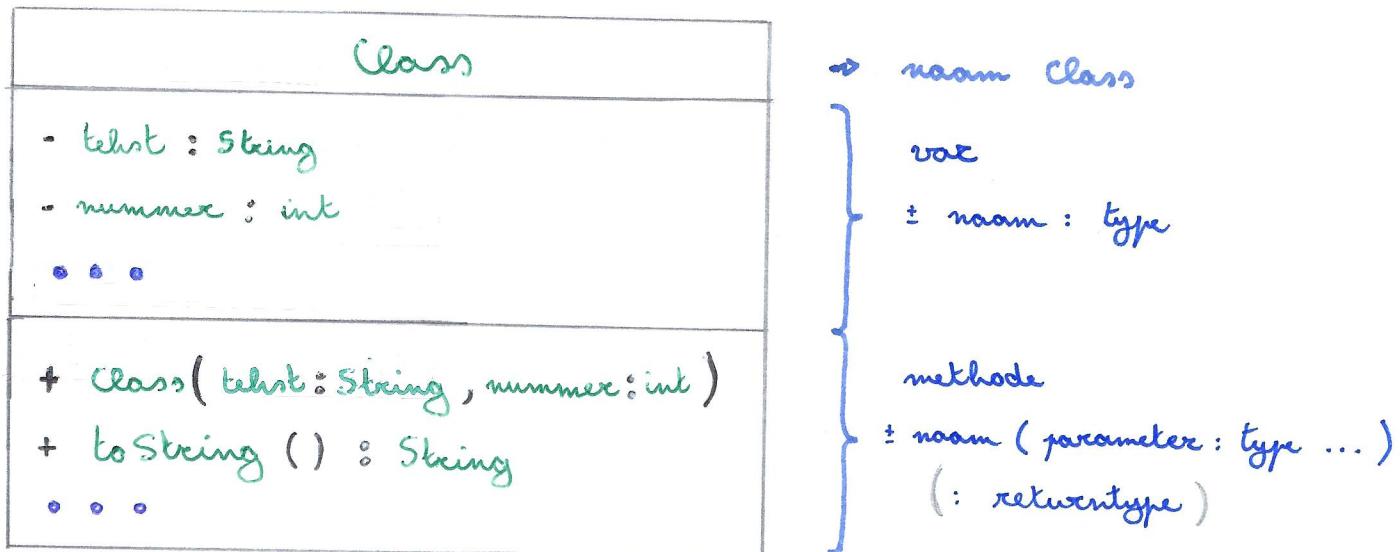


UML



+ public - private # protected

final + static → niet aangegeven

bevat (vac aangegeven door pijl , niet bij vac plaatsen)

→ 1 obj

↳ tenue map

- obj

- ..1 → niet verplicht
- 1 → verplicht

→ collection

- collectie

- 0..* → niet verplicht
- 1..* → verplicht

gebruik (aanspreken maar geen vac)

----->

overerven

----->

abstrakt

→ class

< abstract > Class

→ var → als gewone class

→ methode

< abstract > + methode (): int → herhalen in class die overschrijft

interface (geen var)

→ class

< interface > Class

→ methode → als gewone class → herhalen in class die implementeren

- - - - - ▷

exception

→ class

< exception > Exception

methode throws

+ methode (): int throws Exception

basis

→ class

→ bevat code voor alle obj van dezelfde soort

public class Class { ... }

↳ hoofdletter + hoofdletter voor elk nieuw woord

→ obj

→ class uitvoeren, waardes toekennen aan var

→ veld / attribuut / var

→ eig van obj

↳ eind lijn

private int var (= 12);

↳ kleine letter + hoofdletter voor elk nieuw woord

↳ datatype → primitief / obj (String, Naam...)

→ methode

→ acties op obj, bewerkingen met var

public int methode (int nummer, String tekst...) { ... }

↳ parameters (optioneel)

↳ kleine letter + hoofdletter voor elk nieuw woord

↳ returntype → type var dat wordt teruggeven

→ eind methode

return nummer

→ geen var retournen → void

→ parameter

→ waarde ingevoerd bij uitvoer methode
(int nummer ...)

↳ datatype

→ lokale var

→ var in methode → geld enkel daar
int nummer (= 12);

↳ datatype

class {

var

methodes } }

}

beschikbaarheid

public → overall

leeg → default →zelfde map

protected →zelfde map + subclass

private → class

} opties class

datatype

→ primitief

byte 8 bit

short 16

int 32

long 64

} getal 12

float 32 } hommagetal 12, 12
double 64 }

char 16 → ascii - teken 'A'

boolean 1 → true / false

→ pure bits → geen methodes (equals)

class room → Integer, Double ...

→ methodes

→ obj

String → tekst, array char "tekst"

Class → naam bestaande class - zie oproepen
...

→ casting

getal → getal

(int) nummer lager aantal bits → mogelijk dataverlies

getal → string

String · valueOf (nummer) (" " + nummer)

string → getal

Integer · valueOf (tekst)

speciale methodes

constructor

→ initialiseert obj, waarden toewijzen aan var

public Class (int nummer, String tekst ...) { ... }

↳ parameters (optioneel)

↳ naam class

→ meerdere mogelijk indien aantal parameters verschilt

setter

→ waarde aan var toekennen

public void setNummer (int nummer) { ...

↳ parameter

→ set + naam var (hoofdletter)

nummer = 12 ;

...

↳ parameter / var / lokale var ...

}

getter

→ return var

public int getNummer () { return nummer; }

↳ get + naam var (hoofdletter)

↳ returntype

bewerkingen

voorwaarde

if (a == b) { ... } else { ... }

↳ wanneer false

↳ wanneer true

↳ boolean stelling → true / false

*lyn → geen {}

if (a == b) a++;

else a--;

afhaling

a == b ? a++ : a--;

↳ wanneer false

↳ wanneer true

meerdere voorwaarden met zelfde element

switch (a) {

- case b : a++; \Rightarrow vergelijk a met b
(break;) \Rightarrow indien gelijk voer bewerking
- case c : a--; (a++) uit
(break;)
- default : a++; \Rightarrow indien alle voorwaarde false geven
(optioneel)

→ beindig switch (optioneel)

herhaling → volgens voorwaarde kan herhaal inhoud § 3

→ foc - loop

```
for ( int a = 0 ; a < b ; a++ ) { ... }
```

- ↳ bewerking
- ↳ voorwaarde
- ↳ woe

→ while - loop

while ($a < b$) { ... }

Ld voorwaarde

→ pas op voor infinite loops

rechenregeln

→ rootang

→ wishable (...), *, /, + -

-> links naar rechts

$$S + G + "a" \rightarrow 11a$$

"a" + s + g \rightarrow asg

→ afhalingen

→ $a = a + b \rightarrow a + = b$ werkt met $+, -, *, /$

→ $a * = a + b \rightarrow a = a * (a + b)$

→ $a = a + 1 \rightarrow a ++$
 $- 1 \rightarrow --$

→ functies

$a \% b \rightarrow$ rest deling

Math. abs(a) → absolute waarde a
pow(a, b) → ab

PI → π

sin(a)

cos(a)

logische regels

→ vergelijken

→ getal

$==$ → gelijk

$!=$ → niet gelijk

$< (=)$ → kleiner ($/$ gelijk)

$>$ → groter

→ tekst

a. contains(b) → bevat

• equals(b) → gelijk

→ combineren

&& → and, beide moeten kloppen

|| → or, 1 van beide moet kloppen

tijd + datum

(import java.time.*;) → automatisch boven class

LocalDate date

Time

nu

LocalDate.now()

Time

tekst mitschrijven in terminal

System.out.print("tekst")

println("tekst") → 1 lijn, eindigt met enter

openen

→ class

→ nieuw

Class class = new Class (nummer)

↳ constructor openen

→ bestaand (parameter)

(Class class.)

→ methode

→ intern

(this.) methode (nummer)

→ extern → class openen

class.methode (nummer)

→ var

→ intern

(this.) var

→ extern → class openen

class.var / getter + setter (werkt ook bij private)

recursie

- methode roept zichzelf op
voorwaarde tegen endless loop vereist

ingebouwde functies

klassen hebben overschrijfbare (@Override) ingebouwde functies
veelgebruikte

- equals

@Override

```
public boolean equals ( Obj o ) {
```

```
    if ( o instanceof class ) {
```

```
        Class c = ( Class ) o      → cast
```

```
        if ( ... ) return true    → vergelijk elementen die class
```

```
    }
```

definieeren

```
    return false
```

```
}
```

- toString → teruggeven als string

@Override

```
public String toString () {
```

```
    return "naam: " + a;    → maak string elementen die
```

```
}
```

class definieeren

- comparator → vergelijk 2 obj

→ - < -1

= 0

> 1

- obj moet vergelijkbaar zijn

```
public class Class implements Comparable < Class >
```

@ Override

```
public int compareTo( Obj o ) {
```

... → return int afd van verhouding tot o
}

→ ook mogelijk om class te gebruiken

→ geen implements Comparable < > in obj vergelijkclass

(import java.util.Comparable) automatisch boven class

```
public class Vergelijk implements Comparable< Obj > {
```

@ Override

```
public int compare( Obj a, Obj b ) {
```

... → return int afd van verhouding a <> b
}

}

extra functionaliteit

→ final

```
public final int naam
```

var → niet weigbaar na initiatie (constructor, geen setters)

methode → niet overschrijfbaar door subclass

class → niet extensiebaar

→ static

```
public static int naam
```

→ bruikbaar zonder aanmaken van obj
Class.naam

var →zelfde voor elk obj dat je van class aanmaakt

methode → enkel static var + methodes oplossen

meerdece constructors metzelfde aantal var

```
public static Class a (...) { ... return new Class (...); }  
public static Class b (...) { ... return new Class (...); }  
private Class (...) { ... }
```

Class class = Class a (...) *of* new Class (...)

→ combinatie → etc.

public static final int GETAL-CTE = 7;

speciale class

↳ directe initialisatie

↳ hoofdletters, nieuw woord -

→ subclass

public class Subclass extends Superclass

→ ext eig (var + methode) superclass (generalisatie + specialisatie)

→ in constructor moet superclass worden geinitialiseerd

super (...)

oproepen

→ methode

(super.)methode (...)

↳ verplicht indien subclass methode metzelfde naam bevat

→ var

(super.)var

↳ verplicht indien subclass var metzelfde naam bevat
of

getter + setter (werkt ook bij private)

→ abstract

public abstract class Class

↳ abstract altijd public

→ geen obj aanmaakbaar (nutteloos onder subclass)

abstracte methode (enkel in abstracte class)

public abstract int methode (...); geen body

→ subclass moet deze overschrijven (tenzij zelf abstract)

→ interface

public interface Interface

≈ superclass, maar subclass kan meerdere interfaces implementeren (zelfs in combinatie met superclass)

→ abstract, geen var (behalve cte), geen constructor enkel abstracte methoden

methode → int methode (...);

cte → int CTE = 7

implementeren

public class Sub implements Interface

→ specifieke exception

public class Exception extends IllegalArgumentException {

 public Exception (String s) {

 super (s);

}

}

→ app

public class App {

 public static void main (String [] args) {

 ...

 try { ... } → alles waarbij exception kan optreden

 catch (¹IllegalArgumentException e) {

 (e.getMessage ())

 ...

 }

}

}

→ speciale map

 let nu → src.domain

 app → src.ui

 → moet domein importeren

 import domain.*

→ testen

import org.junit.Test;

 static org.junit.Assert.assertEquals;

public class Test {

 var ...

 @Test ((expected = ¹IllegalArgumentException.class))

 → indien je exception verwacht

 public void methode () {

 (assertEquals (...))

 }

externe bestanden

(import java.io.*;) → automatisch

bij {

File file = new File ("file.txt")

...

↳ in my project

} catch (FileNotFoundException e) { ... }

→ lezen

(import java.util.*;) → automatisch

Scanner scanner = new Scanner (file);

(scanner.nextLine();)

while (scanner.hasNext()) {

hasNext()

scanner.nextLine() ...

next() ...

...

}

→ scanner.useDelimiter (";")

→ splits bij , , onder splitst bij spatie

(splitst altijd bij enter onth van line())

→ schrijven

PrintWriter writer = new PrintWriter (file);

writer.print ("tekst"); → overschrijven
println

close();

array

- collectie van primitieve types of obj
- vaste lengte

int [] array = new int [5] → lengte 5
= {1, 2, 3} → elementen = 1, 2, 3 lengte 3
(= anderetarray → verwijzen naar reflektie adres)

- toewegen + verwijderen → onmogelijk aangerien lengte etc
alternatief → toewegen → nieuw array
verwijderen → null / 0

→ roeden

→ obj o.b.v. index, obj ingeven om te roeden niet mogelijk
array [1] 1ste index = 0, laatste = lengte - 1
aangepassen
array [1] = 12

→ lengte

array.length

2D array

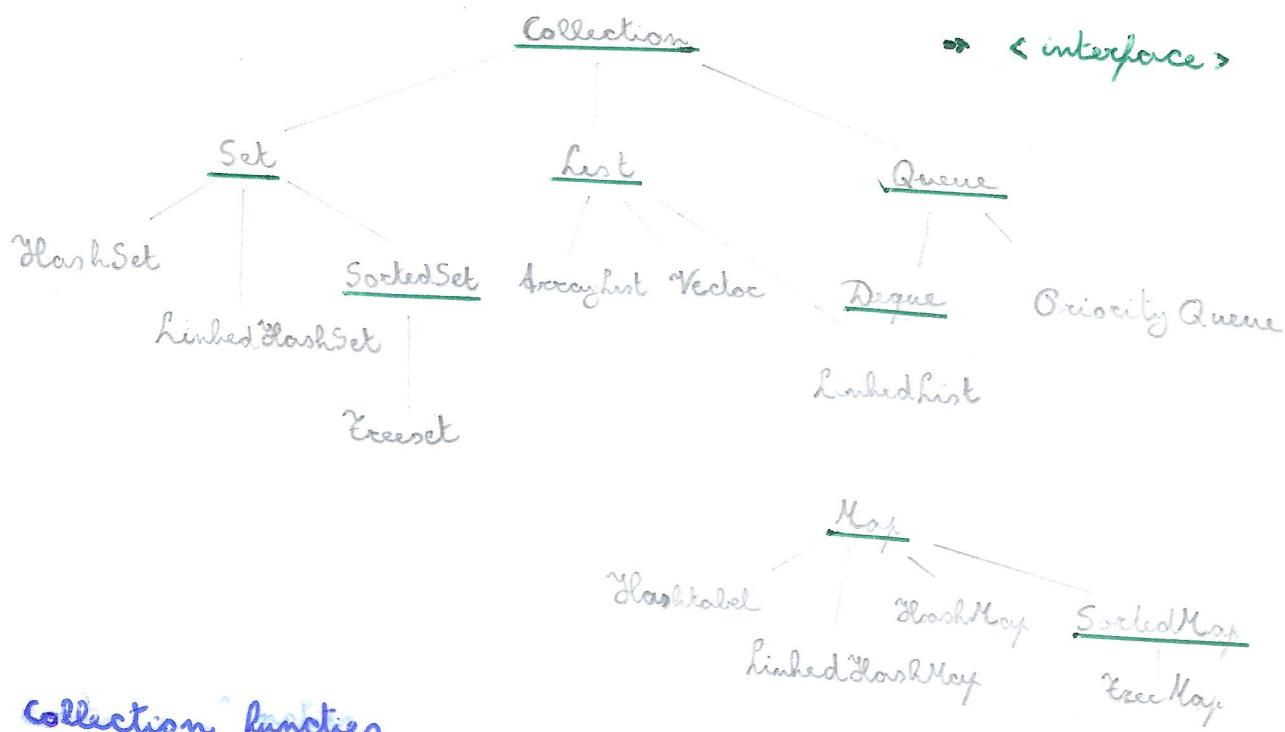
- elementen array = array → 2 indexen

int [][] array = new int [2][3] → lengte 2de var
= {{2, 1}, {4, 3, 5}}
= {eerste, tweede}
(= anderetarray)

length array.length → lengte hoofdarray
[0].length → array op index 0
in hoofdarray

Java Collections FrameWerk

→ interfaces + classes die collecties van obj bijhouden
(java.util.*)



Collection functies

→ toevoegen

lijst.add(obj) : boolean

addAll(anderelijst) : boolean

→ verwijderen

lijst.remove(obj) : boolean

removeAll(anderelijst) : boolean

clear()

→ obov equals()

→ zoeken

lijst.contains(obj) : boolean

containsAll(anderelijst) : boolean

isEmpty() : boolean

→ lengte

lijst. size() : int

→ naar array

lijst. toArray() : obj []

→ iterator

lijst. iterator() : Iterator

List

→ behoud volgorde (achteraan toevoegen)

dubbele + null toegestaan

→ werkt met index

extra functies over index

→ toevoegen

lijst. add(index, obj) : boolean

→ verwijderen

lijst. remove(index) : obj

→ zoeken

lijst. get(index) : obj

indexOf(obj) : int

→ obj.equals()

lastIndexOf(obj) : int

aanpassen (vervangen)

lijst. set(index, obj) : (vervange) obj

→ sorteren

lijst. sort(Comparator)

→ sublijst

lijst. sublist(startindex, endindex) : List
(exclusief) (inclusief)

array list

→ statische rij obj [] van var lengte (initial 10/gehozen)

list < obj > lijst = new arraylist <> ((+12)) \nearrow

zoeken / wijzigen → snel

voegen / verwijderen in midden → traag

linked list

→ pointer 1ste, elk element pointer volgende

list < obj > lijst = new linkedlist <> ();

zoeken / wijzigen → traag

voegen / verwijderen in midden → snel

Queue (wachtrij)

→ behoud volgorde, dubbele + null toegelezen

queue < obj > lijst = new linkedlist <> ();

→ let op, functies list vallen weg

(Queue implementeerd Collection, niet list)

extra functies

→ voegen

lijst.add(obj) : boolean ≈ add (achteraan voegen)

→ verwijderen

lijst.remove() : obj } verwijder 1ste element
poll() : obj }

→ zoeken

lijst.peek() : obj → geef 1ste element terug

Degue (double ended queue)

Degue < obj > lijst = new Degue <> ();

extra functies

→ toevoegen

lijst.addFirst (obj) : boolean
offerFirst (obj) : boolean } voorzaan
addLast (obj) : boolean
offerLast (obj) : boolean } achteraan
push (obj) : boolean → voorzaan

→ verwijderen

lijst.removeFirst () : obj } voorzaan
pollFirst () : obj
removeLast () : obj } achteraan
pollLast () : obj
pop () : obj → achteraan

→ zoeken

lijst.peekFirst () : obj → voorzaan
peekLast () : obj → achteraan

Set

→ geen volgorde, geen dubbelen
(toevoegen dubbele returned false)

Hashset

→ array van elementen (initieel aantal 10 / gehorven → buckets)

Set < obj > set = new HashSet <> ((5));

snel zoeken → index

HashCode → index berekent ob v obj

Hashfunctie → bereken Hashcode

→ voor toevoegen, opvragen, wijzigen

standaard wordt obj memoryadres gebruikt als hashcode
→ hashfunctie in obj toevoegen

(import java.util.Objects;) automatisch boven class

@Override

public int hashCode() {

...

return Objects.hash(nummer, telst);

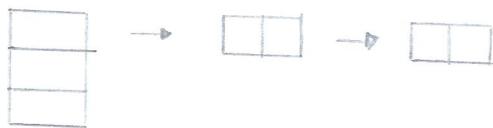
}

↳ var van obj

hashfunctie verschillende obj hanzelfde hashcode behouden

→ collision

→ in linkedlist



↳ buckets

zoeken → key dan linkedlist mbv equals

$$\text{loadfactor} = \frac{\text{aantal entries}}{\text{aantal buckets}}$$

→ loadfactor ↑ → performance may ↓

rehash → verdubbeld aantal buckets en herverdeel entries

goede hashfunctie

→ hashcode groot bereik → large loadfactor

→ gelijkmatige verdeling

Linked HashSet

≈ Hashset, maar behoud volgorde ingevoerd (linkedlist of array)

Set<obj> set = new LinkedHashSet<>((5));

TreeSet

≈ Hashset, maar volgorde volgens orde (comparator)

Set<obj> set = new TreeSet<>((new Vergelijk()));

→ obj moet vergelijkbaar zijn

→ comparators in obj

of

comparatorclass

foreach

→ rechte bewerking voor elk element in collectie

→ collection + array

for (Obj obj : collectie) { bewerking obj ;}

↳ type naam, elementen collectie

lengte van lijst mag niet veranderen bijdens itereren (loop)

opl

→ onderbrek loop na verwijderen

collectie.remove(obj);

break;

→ max 1 element verwijderen

→ iterator

(import java.util.Iterator) automatisch

Iterator i = collectie.iterator();

while (i.hasNext()) {

(Obj obj = i.next();
if (obj ...) { }) → eventuele voorwaarde

i.remove();

(3)

}

Map

→ key-value paar (obj, obj)

→ geen volgorde, geen dubbelen

(toevoegen dubbele returned false)

Map functies

→ toevoegen

map.put(keyObj, valueObj): obj

nieuwe key → null

putAll (andere Map)

bestaande → verwijge obj

→ verwijderen

map.remove(keyObj): obj

remove (keyObj, valueObj): boolean

↳ controleer of valueObj voorkomt
bij keyObj

nee → return false

ja → return true + verwijder

clear()

→ roeden

map.get(keyObj): obj

containsKey(keyObj): boolean

containsValue(valueObj): boolean

isEmpty(): boolean

→ length

map.size(): int

HashMap

→ array key-value paren (initial aantal 10/gehouden → buckets)

Map<keyObj, valueObj> map = new HashMap<>((5));

HashCode → index berekent obv keyObj (zie HashSet)

→ hashfunctie in keyObj toevoegen

LinkedHashMap

≈ Hashmap, maar behoud volgorde ingave (linked list juw array)

Map<keyObj, valueObj> map = new LinkedHashMap<>((5))

TreeMap

≈ Hashmap, maar volgorde volgens order (comparator)

Map<keyObj, valueObj> map = new TreeMap<>((new Vergelijk()));

→ keyObj moet vergelijkbaar zijn

→ compareTo in keyObj

of

comparatorclass

theorie

week 2+3

statische

→ declaratie

@ compile time

dynamisch

→ methode bestaat

→ initialisatie (type constructor)

@ runtime → welke versie

Set = ArraySet

polyfor polymorfisme (overreven + dynamisch)

→ statisch = super

dynamisch = sub

week 4

error

→ throw new exception (...)

→ try { } catch { }

finally { }

→ altijd afhandelen

stop uitvoer

error → buiten applicatie

unchecked → ~~niet~~ niet anticipeerbaar door applicatie

unchecked

→ RuntimeException → binnen applicatie

(programmefout)

niet anticipeerbaar door applicatie

checked

→ Exception

→ anticipeerbaar , gechecked door compiler

→ throws

try { } catch

methode . throws → kan gooien

~~throws~~

done

~~class~~ implements Cloneable
Wiel

@ override

public ~~Object~~ clone() throws CloneNotSupportedException
Wiel
return super.clone()
^
(Wiel)

shallow → ~~alle~~ gebruikte obj refde voor beide
deep → alles uniek

serialize

implements Serializable