

2 Generics & Javadoc

Programmeren 2 – Java

2017 - 2018

KdG Karel de Grote
Hogeschool

Kris Behiels
Jan De Rijke
Mark Goovaerts

Programmeren 2 - Java

1. Herhaling en Collections

2. Generics en documenteren

3. Annotations en Reflection

4. Testen en logging

5. Design patterns (deel 1)

6. Design patterns (deel 2)

7. Lambda's en streams

8. Persistentie (JDBC)

9. XML en JSON

10. Threads

11. Synchronization

12. Concurrency





Generics

Agenda

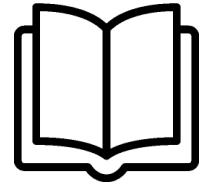
1. Generics

- Voorbeelden
- Generieke methode
- Generieke klasse
- Generieke interface
- Wildcards



2. Documenteren

- Inleiding
- Bestandstypes
- Documentatiecommentaar
- Javadoc tags
- Gebruik van javadoc



E-book: " Generic Classes and Methods" p.49 ev
(Java How to Program, Tenth Edition)

Vóór Java 5: GEEN generics

```
List myList = new ArrayList();  
myList.add("test");
```

Elk object kan
toegevoegd worden

```
String str = (String)myList.get(0);
```

Expliciete cast is nodig!

```
Integer i = (Integer)myList.get(0);
```

Verkeerde cast; wordt **niet** door de
compiler ontdekt en resulteert at
runtime in: `ClassCastException`

Sinds Java 5: generics!



```
List<String> myList = new ArrayList<>();  
myList.add("test");
```

generics en diamond operator
(deze laatste sinds JDK 7)

```
String str = myList.get(0);
```

geen cast nodig!

```
Integer i = myList.get(0);
```

Wordt door de compiler
afgeblokt

Generics: voorbeelden van gebruik




Dit kennen we al:

- Generics bij collections (list, set, ...):

```
List<String> myList = new ArrayList<>();  
myList.add("test");
```

- Generics bij Iterator:

```
Iterator<String> it = myList.iterator();  
while(it.hasNext()) {  
    System.out.println(it.next().toUpperCase());  
}
```



Geen cast nodig; de compiler weet dat het over **String** gaat

Generics: voorbeelden van gebruik



- Generics bij Map (key & value):

```
Map<Klant, List<Bestelling>> myMap = new TreeMap<>();
```

Voor elke klant is er een
List met bestellingen

- Generics bij Comparable:

```
class Klant implements Comparable<Klant>{  
    private int id;  
    private String naam;  
  
    @Override  
    public int compareTo(Klant andereKlant) {  
        return this.id - andereKlant.id;  
    }  
}
```

Generieke methode

- Veronderstel een klasse met overloaded methods:

```
public class OverloadedMethods {  
  
    public static void printArray(Integer[] array) {  
        for(Integer element : array) {  
            System.out.println(element);  
        }  
    }  
  
    public static void printArray(Double[] array) {  
        //...  
    }  
  
    public static void printArray(String[] array) {  
        //...  
    }  
}
```

Hoe vervangen door
1 generieke methode?



Generieke methode

- We vervangen door één generieke methode:

```
public static <E> void printArray(E[] inputArray) {  
    for (E element : inputArray) {  
        System.out.println(element);  
    }  
}
```

```
public static void main(String args[]) {  
    // Create arrays of Integer, Double and String:  
    Integer[] intArray = {1, 2, 3, 4, 5};  
    Double[] doubleArray = {1.1, 2.2, 3.3, 4.4};  
    String[] strArray = {"Just", "Another", "Day"};  
  
    printArray(intArray);  
    printArray(doubleArray);  
    printArray(strArray);  
}
```



Generieke methode: syntax

```
// generic method printArray:  
public static <E> void printArray(E[] inputArray) {  
    for(E element : inputArray) {  
        System.out.println(element);  
    }  
}
```

Type parameter section
Nét voor return type

Meest gebruikte parameter types (naming conventions):

- E – Element (used by the Java Collections)
- K – Key (used in Map)
- N – Number
- T – Type
- V – Value (used in Map)

Je eigen generieke klasse:

de klasse wordt generiek gemaakt voor een bepaald type **T**

```
public class Box<T> {  
    private List<T> myList = new ArrayList<>();  
  
    public void add(T t) {  
        myList.add(t);  
    }  
  
    public T get(int i) {  
        return myList.get(i);  
    }  
  
    @Override  
    public String toString(){  
        StringBuilder sb = new StringBuilder();  
        for(T t : myList) {  
            sb.append(t + " ");  
        }  
        return sb.toString();  
    }  
}
```

Hetzelfde type **T** wordt gebruikt bij de creatie van de List, als parameter, als returntype, ...


Je eigen generieke klasse:

```
public static void main(String[] args) {  
    Box<String> stringBox = new Box<>();  
    stringBox.add("Hello");  
    stringBox.add("World");  
    System.out.println(stringBox);  
}
```

```
Box<Integer> integerBox = new Box<>();  
integerBox.add(10);  
integerBox.add(20);  
System.out.println(integerBox);
```

```
Box generalBox = new Box();  
generalBox.add(5.5);  
generalBox.add("O my God!");  
System.out.println(generalBox);
```

Dit kan ook:
gebruik
maken van
raw types

 Democode:
2_generieke_klasse

```
Hello World  
10 20  
5.5 O my God!
```

Een generieke interface

De Comparable interface:

```
/**
 * ...
 * @param <T> the type of objects that this object may
 * be compared to
 *
 * @author Josh Bloch
 * @see java.util.Comparator
 * @since 1.2
 */
public interface Comparable <T> {
    public int compareTo(T o);
}
```

Bound Generics

? is de **wildcard** in generics

- staat voor: "onbekend type"
- kan gebruikt worden als type voor een parameter, attribuut, lokale variabele of return-waarde

? wordt op 3 manieren toegepast:

- **upper bounded** wildcard: `<? extends Number>`
- **lower bounded** wildcard: `<? super Integer>`
- **unbounded** wildcard: `<?>`

In de upper/lower bound vorm mag je ook een type parameter gebruiken zodat je er naar kan verwijzen: `<N extends Number>`

Upper bound voorbeeld

```
public static double sum(List <? extends Number> list) {  
    double sum = 0;  
    for (Number number : list) {  
        sum += number.doubleValue();  
    }  
    return sum;  
}
```

Methode van de klasse
Number

Number is de upper
bound klasse
LET OP: kan niet
vervallen worden door:
List<Number> list

```
public static void main(String[] args) {  
    List<Integer> ints = new ArrayList<>();  
    ints.add(3); ints.add(5); ints.add(10);  
    double sum = sum(ints);  
    System.out.println("Sum of ints = " + sum);  
}
```

Sum of ints = 18.0
Sum of doubles = 15.0

```
List<Double> doubles = new ArrayList<>();  
doubles.add(1.5); doubles.add(3.5); doubles.add(10.0);  
sum = sum(doubles);  
System.out.println("Sum of doubles = " + sum);
```



Lower bound voorbeeld

```
public static void addIntegers(List <? super Integer> list) {  
    list.add(new Integer(50));  
    list.add(new Integer(100));  
}
```

Alle superklassen van
Integer
Dus: **Integer** is de
lower bound klasse

```
public static void main(String[] args) {  
    List<Object> list1 = new ArrayList<>();  
    addIntegers(list1);  
    System.out.println(list1);  

```

```
List<Integer> list2 = new ArrayList<>();  
addIntegers(list2);  
System.out.println(list2);
```

```
List<Number> list3 = new ArrayList<>();  
addIntegers(list3);  
System.out.println(list3);
```

```
[50, 100]  
[50, 100]  
[50, 100]
```



Unbound voorbeeld

```
public static void printData(List <?> list) {  
    for(Object obj : list){  
        System.out.print(obj + "::");  
    }  
    System.out.println();  
}
```

is hetzelfde als:
<? extends Object>

```
public static void main(String[] args) {  
    List<Integer> ints = new ArrayList<>();  
    ints.add(3); ints.add(5); ints.add(10);  
    printData(ints);  
}
```



Democode: 5_UnBound

```
List<String> strings = new ArrayList<>();  
strings.add("Just"); strings.add("Another"); strings.add("Day");  
printData(strings);
```

```
List<Object> objects = new ArrayList<>();  
objects.add(3.14); objects.add("Hello"); objects.add(new Random());  
printData(objects);  
}
```

3::5::10::

Just::Another::Day::

3.14::Hello::java.util.Random@14ae5a5::

Opdrachten



- Groeiproject

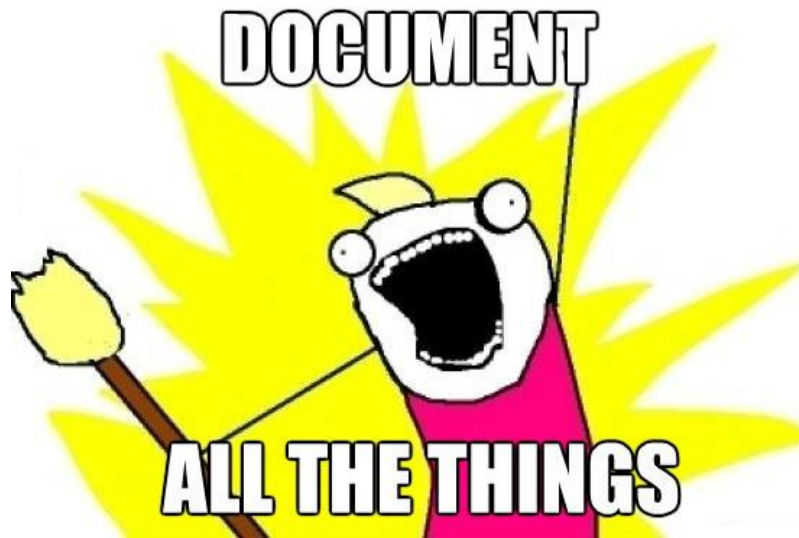
- module 2" Generics en documenteren"
(deel 1 en 2)



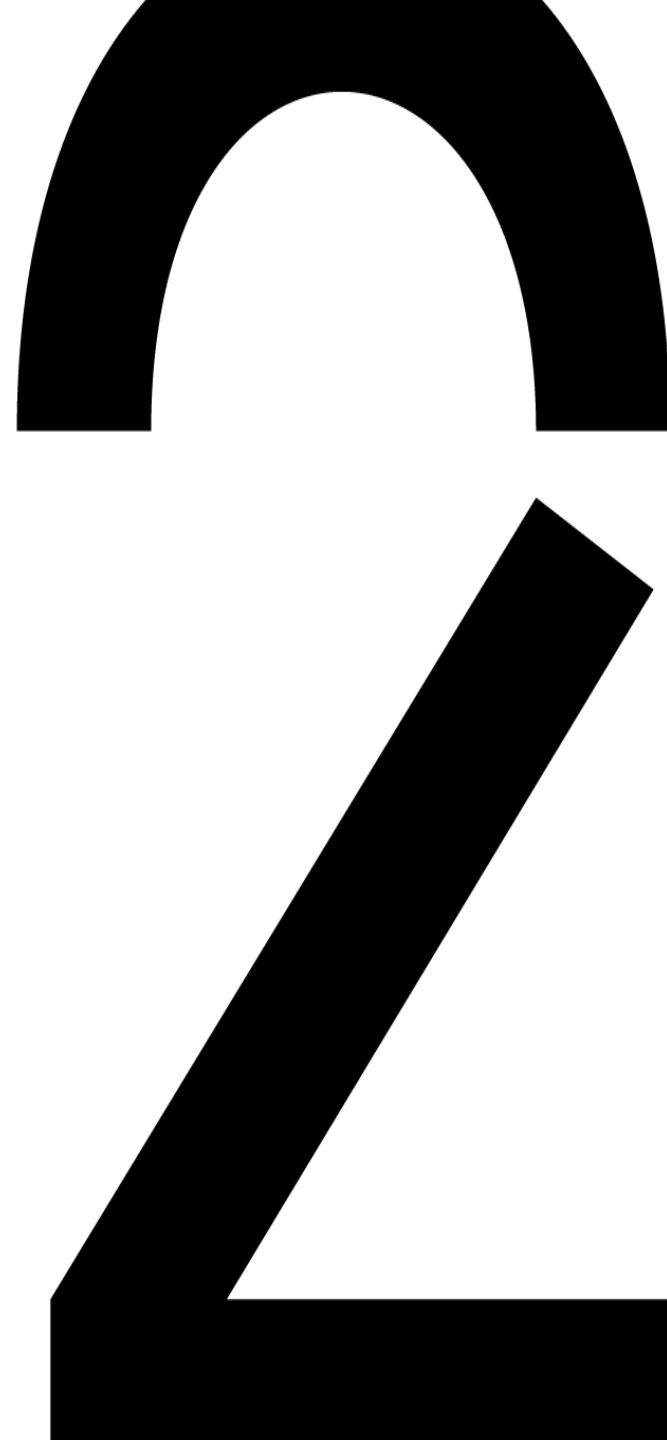
- Opdrachten op BB

- Generic stack
- Generic join
- Generic count





Documenteren



Agenda

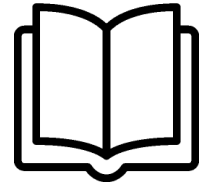
1. Generics

- Voorbeelden
- Generieke methode
- Generieke klasse
- Generieke interface
- Wildcards



2. Documenteren

- Inleiding
- Bestandstypes
- Documentatiecommentaar
- Javadoc tags
- Gebruik van javadoc



E-book hoofdstuk op Blackboard:
"Creating Documentation with javadoc"

Waarom documenteren?

- **Contra:**

- "Goed geschreven code documenteert zichzelf"

- **Pro:**

- Bevordert:

- communicatie tussen informatici
 - onderhoudbaarheid
 - Uitbreidbaarheid

- Brengt structuur in beeld

- (packages, klassen, methoden en hun relaties, ...)

Javadoc

- Javadoc is een **command line tool** uit de JDK die de declaraties en de commentaar in java- bestanden overloopt en daaruit automatisch **HTML documentatie** genereert;
- Javadoc is toepasbaar op volledige packages of op afzonderlijke klassen;
- Javadoc kan doorlinken naar andere HTML pagina's.
- Javadoc is geïntegreerd in vrijwel alle Java IDE's (ook IntelliJ)

Bestandstypes

- Javadoc maakt documentatie vanuit 4 soorten bestandstypes:
 1. Elke *klasse* of *interface* en de *attributen* en *methoden* ervan kunnen documentatie-commentaar hebben
 2. Elke *package* kan een `package.html` bevatten
 3. Elke *verzameling van packages* kan een `overview.html` bevatten
 4. *Andere bestanden* zoals afbeeldingen die je wenst toe te voegen, plaats je in een doc-files map.

Documentatiecommentaar

```
/**
```

```
 * Dit is het klassieke formaat van een documentatie  
 * commentaar over meer dan één regel.
```

```
 */
```

```
/** Dit commentaar staat op een enkelvoudige regel. */
```

```
/*
```

```
 * Dit commentaar wordt NIET gelezen door de Javadoc tool!
```

```
 */
```

```
// Dit commentaar ook NIET!
```

Documentatiecommentaar (2)

Javadoc herkent commentaar alleen wanneer ze **juist boven** de klasse-, interface-, constructor-, methode- of attribuut declaratie staat!

Het volgende is dus fout:

```
/**
 * Commentaar voor de klasse Foo
 */
import javax.swing.*;

public class Foo {
    // . . .
}
```

Onderverdeling documentatiecommentaar

- Een documentatiecommentaar bestaat uit een **beschrijving** gevolgd door een **tag gedeelte**.

```
/**
```

```
{ * Maakt een nieuw Vierkant object.  
  *  
  { * @param zijde De zijde van het vierkant  
    * @return Een Vierkant object  
  */
```

- Een tag is een speciale **annotation** in een commentaarzone, die voor de Javadoc tool een speciale betekenis heeft.

Over annotations zullen we het volgende week uitgebreid hebben!

HTML-tags

- ook **html-tags** worden aanvaard:

```
/**  
 * Maakt een <b>nieuw</b> Session object.  
 *  
 * @param status <i>True</i> voor een geldige sessie,  
 *      <i>false</i> voor een ongeldige.  
 */  
public void newSession(boolean status) {
```

HTML-tags

- ook **html-tags** worden aanvaard:

```
/**
```

```
 * Maakt een <b>nieuw</b> Session object.
```

```
 *
```

```
 * @param status <i>True</i> voor een geldige sessie,
```

```
 *   <i>false</i> voor een ongeldige.
```

```
 */
```

```
public void newSession(boolean status)
```

Method Detail

newSession

```
public void newSession(boolean status)
```

Maakt een **nieuw** Session object.

Parameters:

status - *True* voor een geldige sessie, *false* voor een ongeldige.

Resultaat in HTML:

<pre> tag

- Voorafgaande asterisken worden door de Javadoc parser genegeerd
- insprongen: door **<pre>** tags in je commentaar te plaatsen.

```
/**
 * Klasse FiguurFactory.
 * Factory voor figuur-objecten.
 *
 * <pre>
 *   Bijvoorbeeld:
 *
 *       Figuur vk = FiguurFactory.getVierkant(4.0),
 *       Figgur rh = FiguurFactory.getRechthoek(3.0, 2.0),
 *       Figuur fg = FiguurFactory.getFiguur(FiguurType.VIERKANT, 4.0)
 * </pre>
 */
```


<pre> tag

- Voorafgaande asterisken worden door de Javadoc parser genegeerd
- insprongen: door **<pre>** tags in je commentaar te plaatsen.

```
/**
 * Klasse FiguurFactory.
 * Factory voor figuur-objecten.
 *
 * <pre>
 *   Bijvoorbeeld:
 *
 *       Figuur vk = FiguurFact
 *       Figgur rh = FiguurFact
 *       Figuur fg = FiguurFact
 * </pre>
 */
```

Resultaat in HTML:

```
be.kdg.junitVb
Class FiguurFactory
java.lang.Object
└─ be.kdg.junitVb.FiguurFactory
```

```
public class FiguurFactory
extends java.lang.Object
```

Klasse FiguurFactory. Factory voor figuur-objecten.

Bijvoorbeeld:

```
Figuur vk = FiguurFactory.getVierkant(4.0),
Figgur rh = FiguurFactory.getRechthoek(3.0, 2.0),
Figuur fg = FiguurFactory.getFiguur(FiguurType.VIERKANT, 4.0)
```

Beschrijving

- De eerste zin = korte **beschrijving** van wat de klasse, interface, methode, ... doet.
- Javadoc kopieert deze zin in de samenvatting bovenaan de gegenereerde html pagina.

```
/**  
 * Maakt een nieuwe Figuur waarbij de twee dimensies gelijk zijn.  
 * De figuur is ofwel een Vierkant, ofwel een Rechthoek waarbij  
 * breedte en hoogte gelijk zijn.  
  
 * @param type Het type figuur  
 * @param zijde De afmeting van de figuur  
 * @return Een nieuw Vierkant of Rechthoek object.  
 */
```

Beschrijving

- De eerste zin = korte **beschrijving** van wat de klasse, interface, methode, ... doet.
- Javadoc kopieert deze zin in de samenvatting bovenaan de gegenereerde html pagina.

```
/**  
 * Maakt een nieuwe Figuur waarbij de twee dimensies gelijk zijn.  
 * De figuur is ofwel een Vierkant, ofwel een Rechthoek waarbij  
 * breedte en hoogte gelijk zijn.  
  
 * @param type Het type figuur  
 * @param zijde De afmeting van  
 * @return Een nieuw Vierkant of  
 */
```

maakFiguur

```
public Figuur maakFiguur(int type,  
                        double zijde)
```

Maakt een nieuwe Figuur waarbij de twee dimensies gelijk zijn. De figuur is ofwel een

Parameters:

type - Het type figuur
zijde - De afmeting van de figuur

Returns:

Een nieuw Vierkant of Rechthoek object.

Resultaat in HTML:

Javadoc tags

- zijn annotations met een speciale betekenis binnen de commentaar;
- beginnen altijd met een @-teken;
- zijn "case sensitive";
- moeten aan het begin van een regel staan
- bestaan in twee vormen:
 - **Block** tags:
van de vorm @tag, je plaatst ze na de algemene beschrijving
 - **Inline** tags:
van de vorm {@tag}, je kan ze overal in de commentaar plaatsen

Javadoc tags

• Block tags

- @author
- @deprecated
- @exception
- @param
- @return
- @see
- @serial
- @serialdata
- @serialfield
- @since
- @throws
- @version

• Inline tags

- {@docRoot}
- {@inheritDoc}
- {@link}
- {@value}

Javadoc block tags

- **@author**

- Voegt een author gedeelte toe wanneer de `-author` optie aan staat.

```
/**  
 * @author Kristiaan Behiels, Mark Goovaerts  
 */
```

```
/**  
 * @author Kristiaan Behiels  
 * @author Mark Goovaerts  
 */
```

Javadoc block tags

- **@deprecated**

- Geeft aan dat je dit API gedeelte niet meer mag gebruiken.
- Geef zeker een link naar de nieuwe methode.

```
/**
 * Geeft aan of de sessie geldig is.
 * @return true bij een geldige sessie, anders false
 * @deprecated Sinds versie 1.1, gebruik de
 *             {@link #isGeldig() isGeldig} methode.
 */
public boolean geldig() {
    return geldig;
}
```

Javadoc block tags

- **@deprecated**

- Geeft aan dat je dit API gedeelte niet meer mag gebruiken.
- Geef zeker een link naar de nieuwe methode.

```
/**
 * Geeft aan of de sessie geldig is.
 * @return true bij een geldige sessie, anders false
 * @deprecated Sinds versie 1.1, gebruik de
 *             {@link #isGeldig()}
 */
public boolean geldig()
    return geldig;
}
```

Resultaat in HTML:

Method Detail

geldig

```
public boolean geldig()
```

Deprecated. Sinds versie 1.1, gebruik de [isGeldig](#) methode.
Geeft aan of of de sessie geldig is.

Returns:

true bij een geldige sessie, anders false

Javadoc block tags

- **@exception**

- Synoniem voor @throws

- **@param**

- Voegt een parameter toe aan het parameter gedeelte, alleen mogelijk bij methoden (of constructors)

```
/**
```

```
 * Constructor voor een Datum-object op basis van dag, maand en jaar
```

```
 * @param dag De dag
```

```
 * @param maand De maand
```

```
 * @param jaar Het jaar
```

```
 */
```

```
KdG public Datum(int dag, int maand, int jaar) {
```

```
    // ...
```

Javadoc block tags

- **@exception**

- Synoniem voor @throws

- **@param**

- Voegt een parameter toe aan het parameter gedeelte, alleen mogelijk bij methoden (of constructors)

```
/**  
 * Constructor voor een Datum  
 * @param dag De dag  
 * @param maand De maand  
 * @param jaar Het jaar  
 */
```

```
public int
```

Resultaat in HTML:

```
//...
```

Constructor Detail

Datum

```
public Datum(int dag,  
             int maand,  
             int jaar)
```

Constructor voor een Datum-object op basis van dag, maand en jaar

Parameters:

dag - De dag
maand - De maand
jaar - Het jaar

Javadoc block tags

- **@return**

- Voegt een return gedeelte toe met een verklarende tekst, alleen geldig bij methoden.

```
/**
 * Test of een datum mogelijk is.
 * Datums voor het jaar 1600 worden als fout beschouwd.
 * @param dag De dag
 * @param maand De maand
 * @param jaar Het jaar
 * @return true bij een geldige datum, anders false
 */
public static boolean isGeldigeDatum (int dag, int maand,
                                     int jaar) {
```

Javadoc block tags

- **@return**

- Voegt een return gedeelte toe met een verklarende tekst, alleen geldig bij methoden.

```
/**
```

```
 * Test of een datum mogelijk is.
```

```
 * Datums voor het jaar 1600 worden als fout beschouwd
```

```
 * @param dag De dag
```

```
 * @param maand De maand isGeldigeDatum
```

```
 * @param jaar Het jaar public static boolean isGeldigeDatum(int dag,
```

```
 * @return true bij een int maand,
```

```
 */
```

```
public static boolean
```

Resultaat in HTML:

Method Detail

isGeldigeDatum

```
public static boolean isGeldigeDatum(int dag,  
                                     int maand,  
                                     int jaar)
```

Test of een datum mogelijk is. Datums voor het jaar 1600 worden als fout beschouwd

Parameters:

dag - De dag
maand - De maand
jaar - Het jaar

Returns:

true bij een geldige datum, anders false

Javadoc block tags

- **@see**

- Voegt een "see also" hoofding met een verwijzing toe.
Een commentaar mag meer dan een see-tag bevatten.
- De see-tag heeft drie varianten:
 - **@see "string" '**
 - Voor informatie zonder link
 - **@see label**
 - Voegt een link toe, URL#value is een relatieve of absolute URL.
 - **@see package.class#member label**
 - Voegt een link toe naar de aangegeven package, class of member.
 - Alleen *label* is zichtbaar in de tekst

Javadoc block tags

- **@see**

```
/**  
 * @see <a href="spec.html#section">JavaSpec</a>  
 * @see String#equals(Object) The String equals method  
 * @see java.lang.Object#equals equals  
 */
```

- **@serial**

- **@serialField**

- **@serialData**

– Tags voor het beschrijven van persistente (serializable) members.

Javadoc block tags

- **@since**

- Aanduiding vanaf welke release (version) de package, class, interface of method bestaat.

`@since 1.4`

- **@version**

- Aanduiding van de huidige versie (indien de `-version` optie wordt gebruikt).

`@version 1.5, 11 okt 2016`

Javadoc block tags

- **@throws**

- Alleen toegelaten bij een constructor of een methode
- @throws en @exception zijn synoniemen

```
/**
```

```
 * Geeft een bepaald element van de tabel terug.  
 * @param index Positie van het element in de tabel  
 * @return Een willekeurig getal van het type double  
 * @throws IndexOutOfBoundsException  
 */
```


Javadoc inline tags

- **{@docroot}**

- Stelt het relatieve path naar de gegenereerde documentatie voor. Nuttig voor het toevoegen van een bestand zoals een copyright pagina of een logo. Deze tag is geldig in alle documenten.

```
/**  
 *   Zie <a href="{@docroot}/copyright.html">Copyright</a>  
 */
```

- Meer info over tags:
http://www.tutorialspoint.com/java/java_documentation.htm

Aanbevolen tag volgorde

- * @author
- * @version (alleen klassen en interfaces)
- * @param (alleen methoden en constructors)
- * @return (alleen methoden)
- * @exception (of @throws - is synoniem)
- * @see
- * @since
- * @serial (of @serialField of @serialData)
- * @deprecated

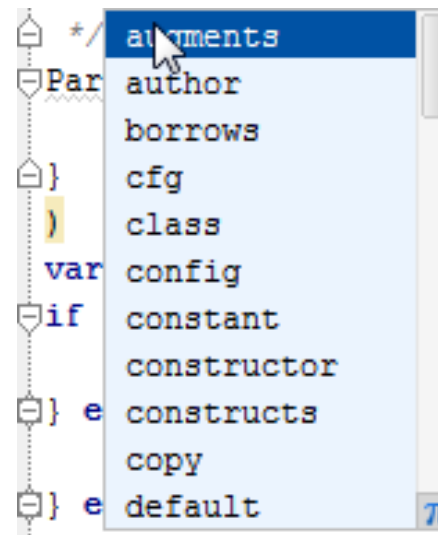
Hoe javadoc toevoegen in IntelliJ?

/** <ENTER>

–IntelliJ vervolledigt, inclusief (lege) tags, voorbeeld:

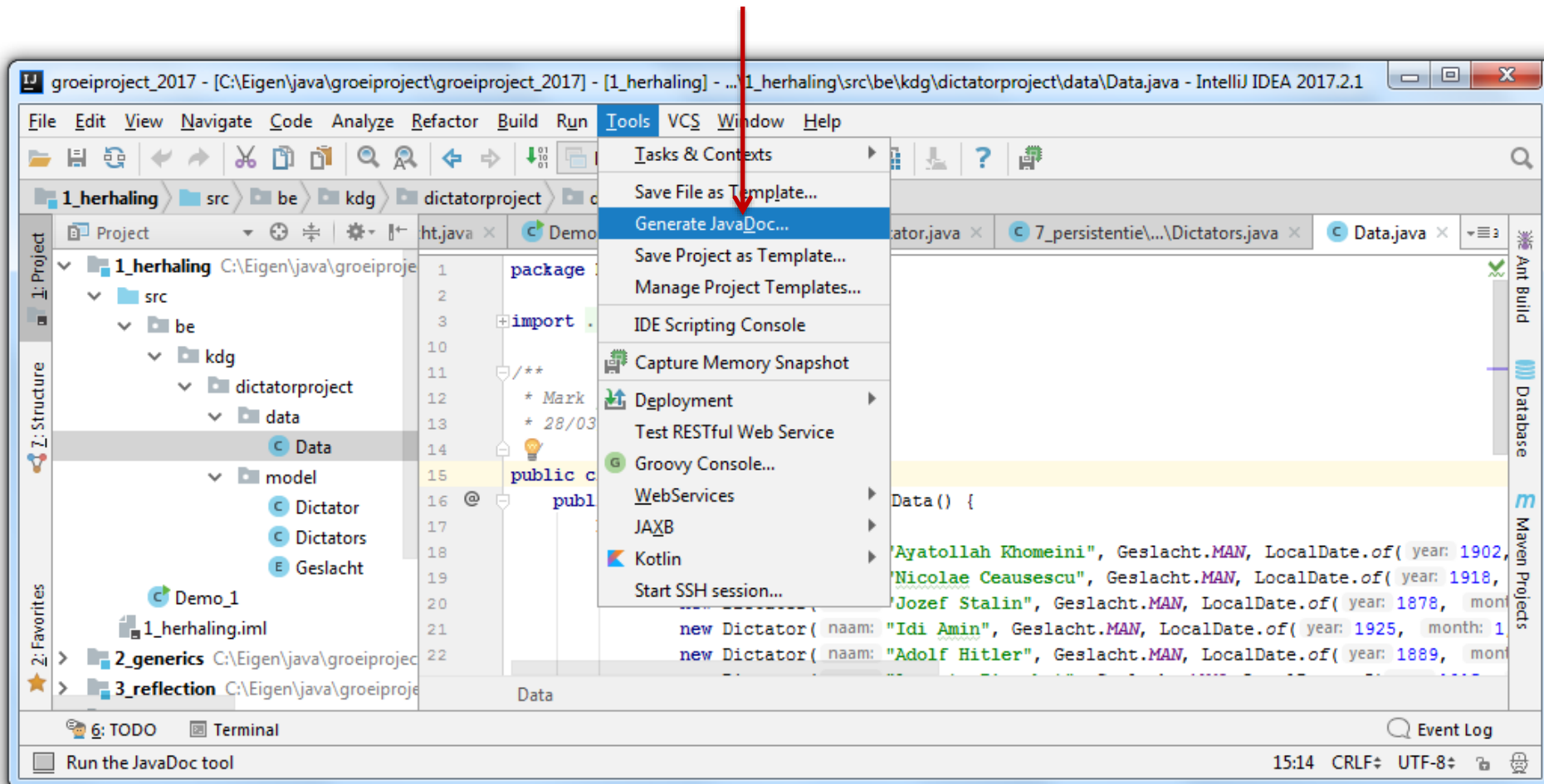
```
/**
 * @param type
 * @param zijde
 * @return
 */
public Figuur maakFiguur(int type, double zijde)
```

–Uiteraard is er ook autocomplete voor tags. @<CTRL><SPACE>:



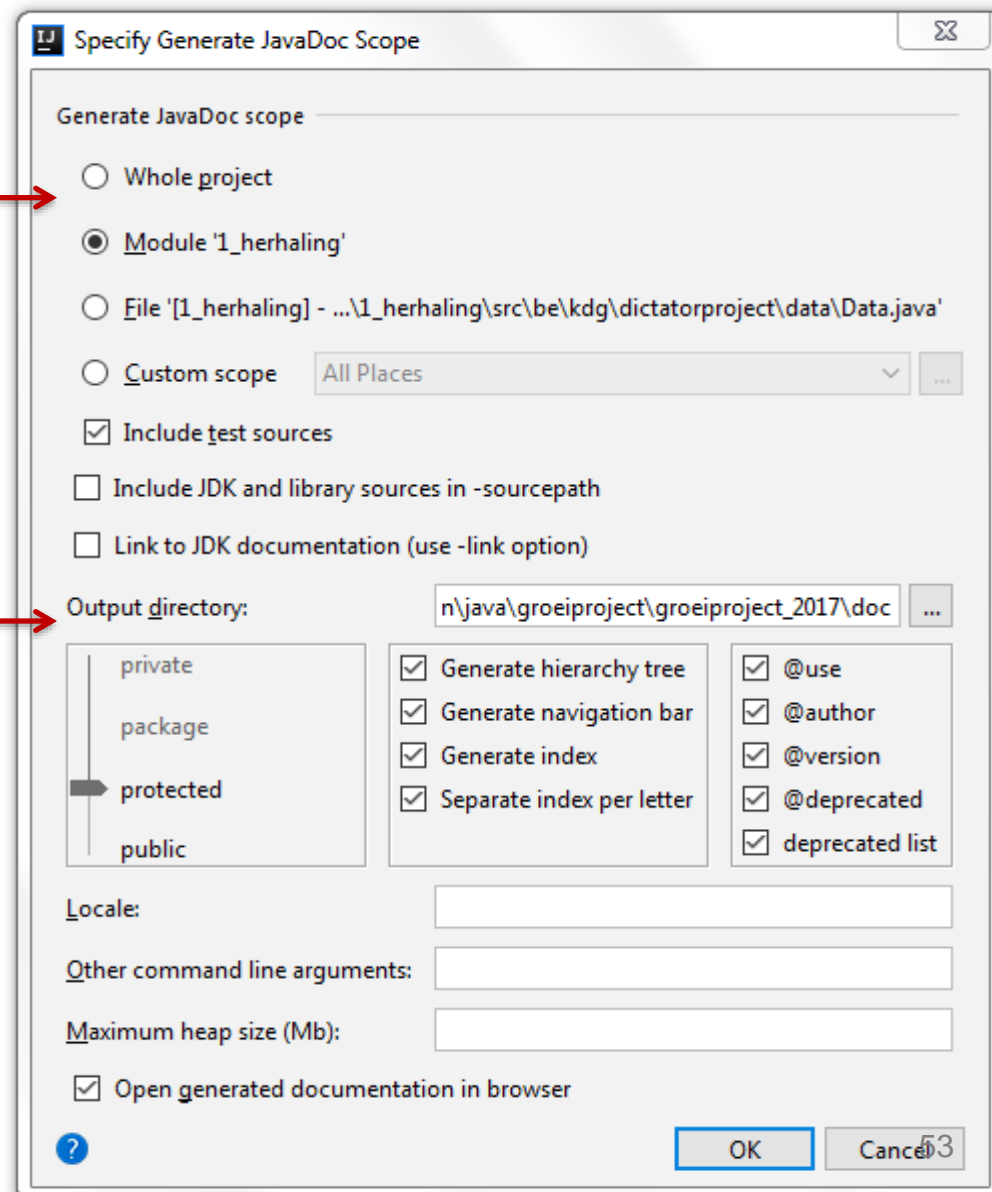
Hoe javadoc genereren vanuit IntelliJ?

- Kies *Generate JavaDoc...* in het Tools menu



Hoe javadoc genereren vanuit IntelliJ?

- Kies de **scope**: "Whole project" of "Module" —→
- Stel de **output directory** in (kies voor een directory *doc* op het niveau van de *src* directory) —→
- Klik vervolgens op "**OK**" en bekijk de documentatie in je browser



Democode

- De voorbeelden uit de voorgaande slides:



Democode: 6_Javadoc

- Een voorbeeld uit de praktijk, met name de broncode van de klasse **Assert** uit JUnit. Hierbij ook de gegenereerde documentatie in de map doc.



Democode: 7_Klasse_Assert


- Merk op dat Javadoc nog steeds HTML4.01 code genereert! (HTML5 is gepland voor JDK9)

Gebruik van de javadoc commandline instructie

javadoc [options] [packagenames] [sourcefilenames]

[-subpackages *pkg1:pkg2:...*] [@argfiles]

javadoc.exe vind
je in de JDK-
folder\bin



- options: Command-line opties
- packagenames: Een reeks namen van packages, gescheiden door spaties (geen wildcards). Javadoc gebruikt –sourcepath om de packages te zoeken.
- sourcefilenames: Een reeks namen, gescheiden door spaties (path en wildcards toegestaan)
- -subpackages *pkg1:pkg2:...* Maakt documentatie voor de broncode in de opgegeven packages en recursief in hun subpackages
- @argfiles: Een of meer bestanden die een lijst van javadoc options, packagenamen of broncodenamen bevatten (geen wildcards)

Gebruik van de javadoc commandline instructie

```
C:\>javadoc -d H:\java\doc -sourcepath H:\java\src -subpackages be
```

Documenteert alles in de package `be` (locatie `H:\java.src\`) en de subpackages ervan. De documentatie wordt in de `H:\java\doc` directory geplaatst.

```
C:\>cd H:\java\src
```

```
H:\java\src>javadoc -d H:\java\doc demo be.kdg.util be.kdg.factory
```

Documenteert alles in de packages `demo`, `be.kdg.util` en `be.kdg.factory` (locatie `H:\java.src\`). De documentatie wordt in de `H:\java\doc` directory geplaatst.

```
H:\java\src>javadoc -d ..\doc -overview overview.html be.kdg.factory
```

Documenteert alles in de package `be.kdg.factory`. De documentatie wordt in de `H:\java\doc` directory geplaatst. Het bestand `overview.html` in de huidige directory wordt mee opgenomen.

Opdrachten



- Groeiproject

- module 2" Generics en documenteren"
(deel 3: Javadoc)



- Opdrachten op BB

- Generic stack
- Generic join
- Generic count

