

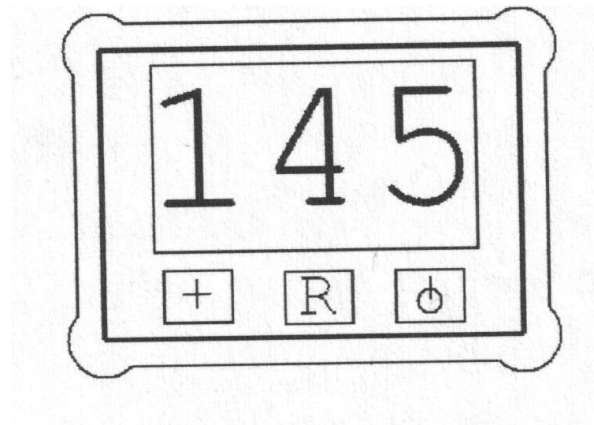
Odissee  
DE CO-HOGESCHOOL

# UML en Documentatie



Maarten Troost – Jens Baetens

## Hoe beschrijf je een applicatie?



## Methode 1: Beschrijving in taal

- ▣ Een teller dient om het aantal malen van een bepaalde gebeurtenis bij te houden.
- ▣ Hij kan in- en uitgeschakeld worden.
- ▣ Indien een teller ingeschakeld is, dan bezit hij de mogelijkheid de huidige stand mee te delen.
- ▣ Nog steeds als hij ingeschakeld is, kan de teller met 1 verhoogd worden.
- ▣ Hij kan ook terug op 0 gezet worden.
- ▣ Indien een teller uitgeschakeld is, geeft de display steeds een 0 weer en reageren de knoppen om te resetten en de waarde te verhogen niet.



## Methode 2: Grafische beschrijving

### ▣ Bijvoorbeeld via UML

- Unified Modelling Language

Teller	
-	waarde: int
-	tellerStaatAan: boolean
+	Teller()
+	increment(): void
+	reset(): void
+	getWaarde(): int
+	zetTellerAan(): void
+	zetTellerUit(): void
+	staatTellerAan(): boolean

## Methode 3: Beschrijving in programmeertaal

```
package be.odisee;

/**
 * Een eenvoudige teller
 * @author Jens Baetens
 */
public class Teller {

    private int waarde;
    private boolean tellerStaatAan;

    /**
     * Default constructor: een Teller-objekt wordt aangemaakt
     * met waarde 0 en is uitgeschakeld.
     */
    public Teller(){
        this.waarde = 0;
        this.tellerStaatAan = false;
    }

    /**
     * Vermeerder waarde van Teller met 1.
     * Heeft geen effect als de teller uit staat.
     */
    public void increment(){
        if (tellerStaatAan){
            this.waarde++;
        }
    }
}
```

```
/**
 * Zet waarde van Teller op 0.
 * Heeft geen effect als de teller uit staat.
 */
public void reset(){
    if (tellerStaatAan){
        this.waarde = 0;
    }
}

/**
 * Return waarde van de Teller
 * Indien de teller uitstaat, geef nul terug.
 */
public int getWaarde(){
    if (tellerStaatAan) {
        return this.waarde;
    } else {
        return 0;
    }
}

/** Zet teller aan en zet waarde op 0 */
public void zetTellerAan(){
    this.tellerStaatAan = true;
    this.waarde = 0;
}

/** Zet teller uit en zet waarde op 0 */
public void zetTellerUit(){
    this.tellerStaatAan = false;
    this.waarde = 0;
}

/** Controleer of de teller aanstaat */
public boolean staatTellerAan(){
    return this.tellerStaatAan;
}
```

## Hoe deze klasse Teller gebruiken?

- ▣ Maakt Teller-object aan
- ▣ Zet de teller aan
- ▣ Duw drie keer op de knop om te incrementeren
- ▣ Lees de waarde uit
- ▣ Reset
- ▣ Lees opnieuw de waarde uit
- ▣ Zet de teller uit

```
public static void main(String[] args) {  
  
    Teller teller = new Teller();  
  
    teller.zetTellerAan();  
  
    if(teller.staatTellerAan()){  
        teller.increment();  
        teller.increment();  
        teller.increment();  
  
        System.out.println("Huidige waarde van teller " + teller.getWaarde());  
  
        teller.reset();  
  
        System.out.println("Huidige waarde van teller " + teller.getWaarde());  
    }  
  
    teller.zetTellerUit();  
}
```



# Project Lifecycle



## Wat wil dit zeggen voor de project lifecycle?

- ▣ Doorgronden van de probleemstelling
- ▣ Opstellen van lijst met te leveren publieke diensten
- ▣ Herhaal:
  - ▬ JUnit test maken (op basis van de lijst)
  - ▬ Schrijf code (eerst het skelet, daarna de details)
  - ▬ Refactor
  - ▬ Javadoc-documentatie schrijven
- ▣ En daarna ... de GUI dus

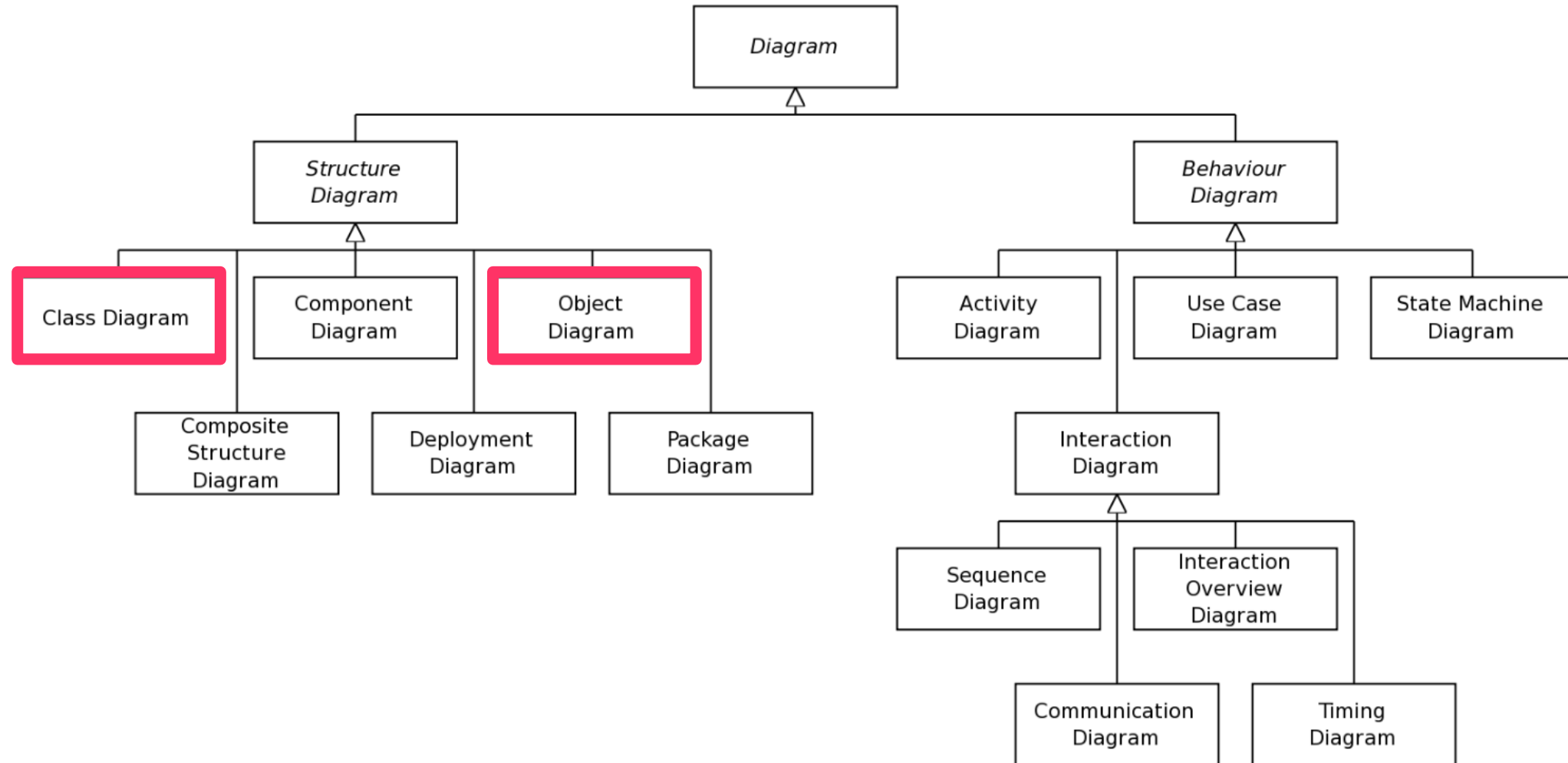


# UML

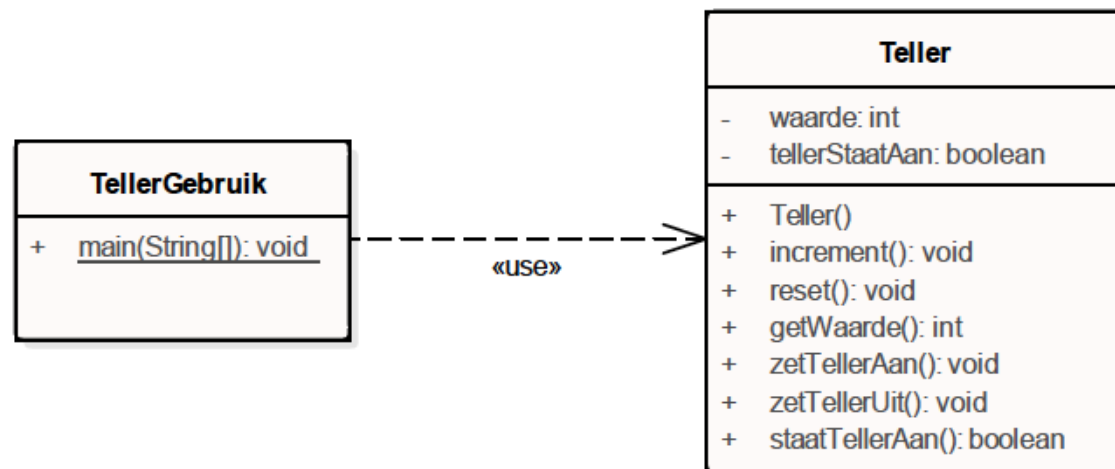
# UML of Unified Modelling Language

- ▣ Set van symbolen en syntaxregels om software grafisch voor te stellen
- ▣ Tussen natuurlijke taal en code
- ▣ Voordelen
  - Gedachten ordenen
  - Discussiëren over de architectuur/ontwerp
  - Aftoetsen van volledigheid/kwaliteit: Is er aan alles gedacht? Wat ontbreekt?
  - Complexiteit reduceren en overzicht behouden
- ▣ Er bestaan tools om UML in code om te zetten en omgekeerd

# Soorten UML-diagrammen



# Het UML – klassendiagram van de Teller applicatie



# Schema van een klasse



# Attributen / Eigenschappen / Properties

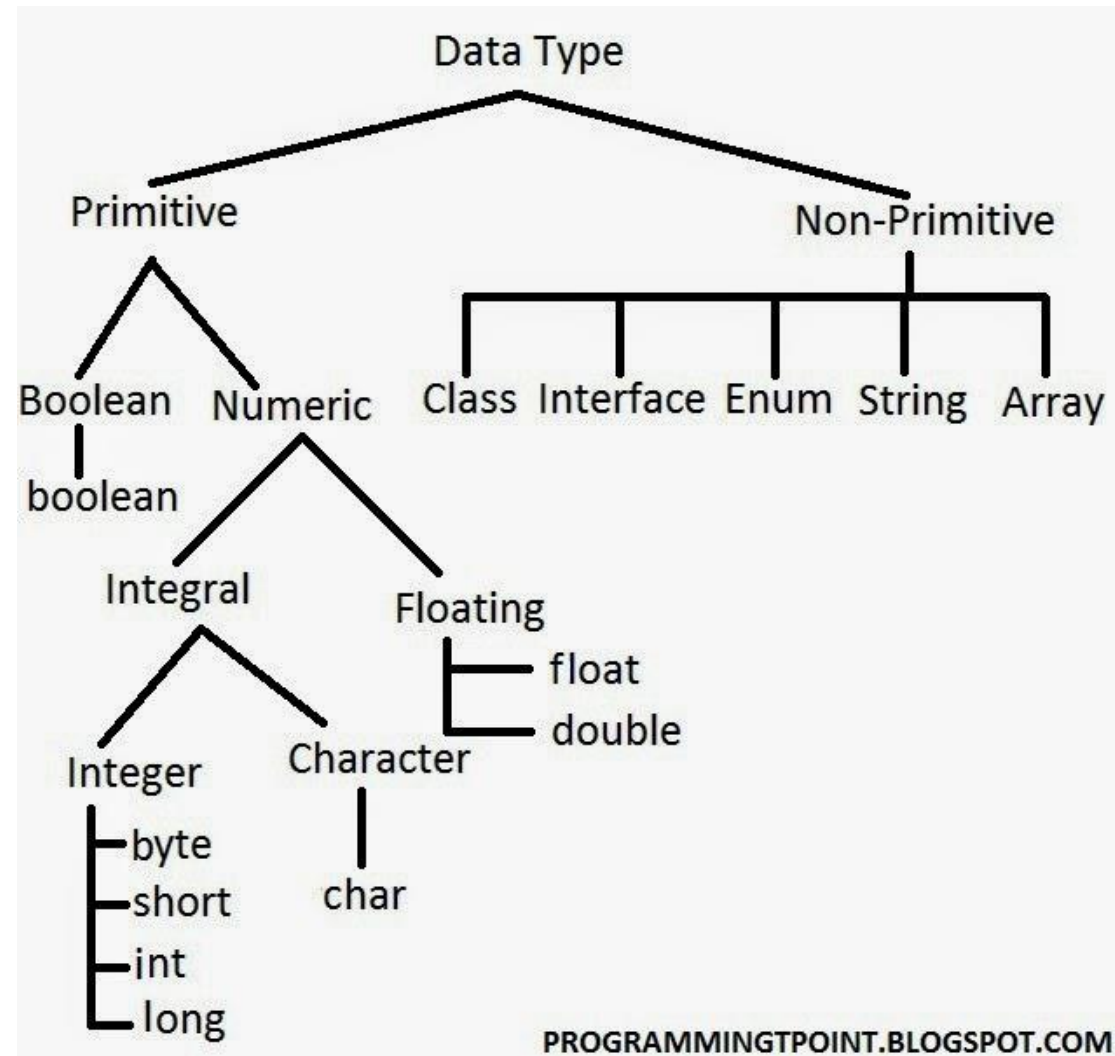
Teller	
-	waarde: int
-	tellerStaatAan: boolean
+	Teller()
+	increment(): void
+	reset(): void
+	getWaarde(): int
+	zetTellerAan(): void
+	zetTellerUit(): void
+	staatTellerAan(): boolean

```
public class Teller {  
  
    private int waarde;  
    private boolean tellerStaatAan;  
}
```

Elk attribuut heeft 3 kenmerken:

1. **Zichtbaarheid**
  - voor private
  - + voor public
  - # voor protected
2. **Type**
  - Type of klasse van de variabele
3. **Naam**
  - begint met kleine letter heeft betekenis







## Welke sleutelwoorden zijn er

### ▣ Voor visibility

- ▬ public
- ▬ private
- ▬ protected
- ▬ package

### ▣ static

- ▬ Property gedeeld door alle objecten in de klasse
- ▬ Property van de klasse, niet van het object

### ▣ final

- ▬ Voor constanten
- ▬ Kan enkel in de constructor ingesteld worden

# Methodes

Teller
- waarde: int - tellerStaatAan: boolean
+ Teller() + increment(): void + reset(): void + getWaarde(): int + zetTellerAan(): void + zetTellerUit(): void + staatTellerAan(): boolean

Elke methode heeft 3 kenmerken:

1. **Zichtbaarheid**

- voor private
- + voor public
- # voor protected

2. **Return type**

Type of klasse van de return-waarde

3. **Naam**

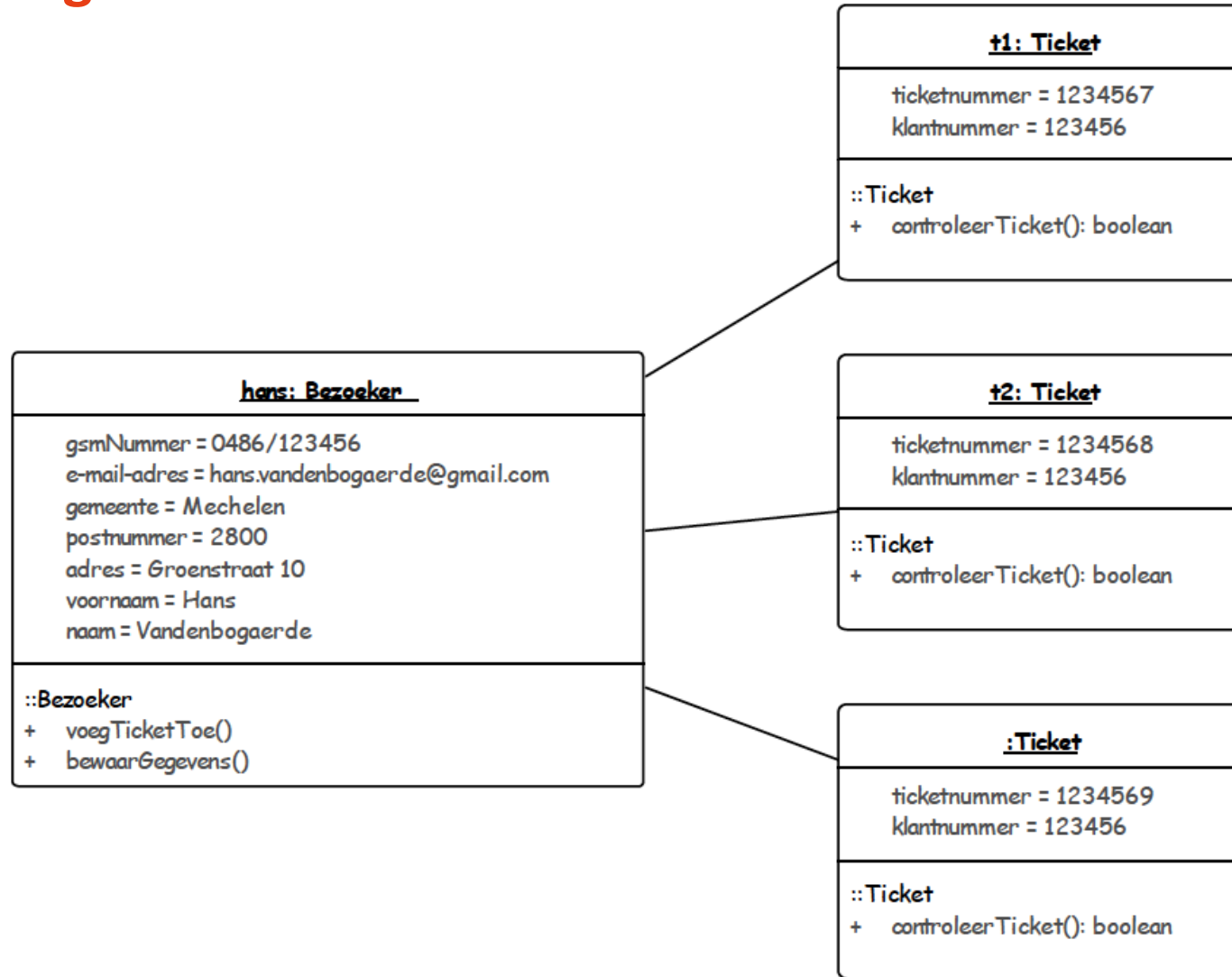
begint met kleine letter behalve de constructor

```
public Teller(){
    this.waarde = 0;
    this.tellerStaatAan = false;
}

/**
 * Vermeerder waarde van Teller met 1.
 * Heeft geen effect als de teller uit staat.
 */
public void increment(){
    if (tellerStaatAan){
        this.waarde++;
    }
}
```

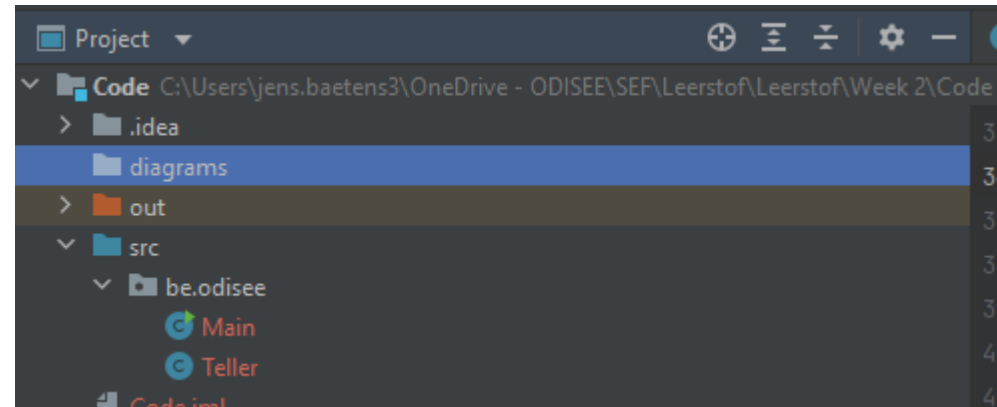
```
public int getWaarde(){
    if (tellerStaatAan) {
        return this.waarde;
    } else {
        return 0;
    }
}
```

# UML – object diagram

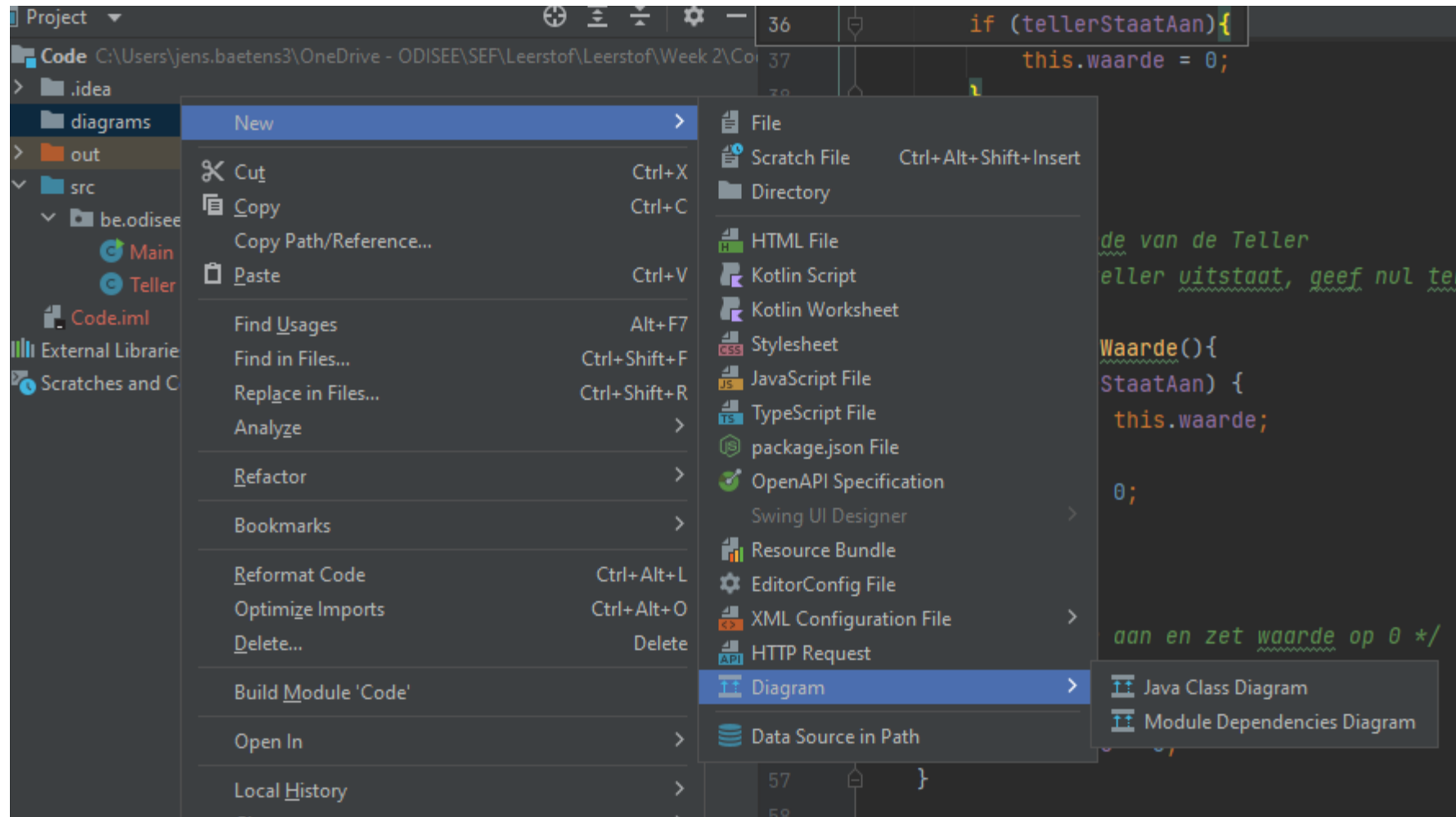


## UML in IntelliJ

- ▣ Maak een project aan
- ▣ Voeg een folder toe



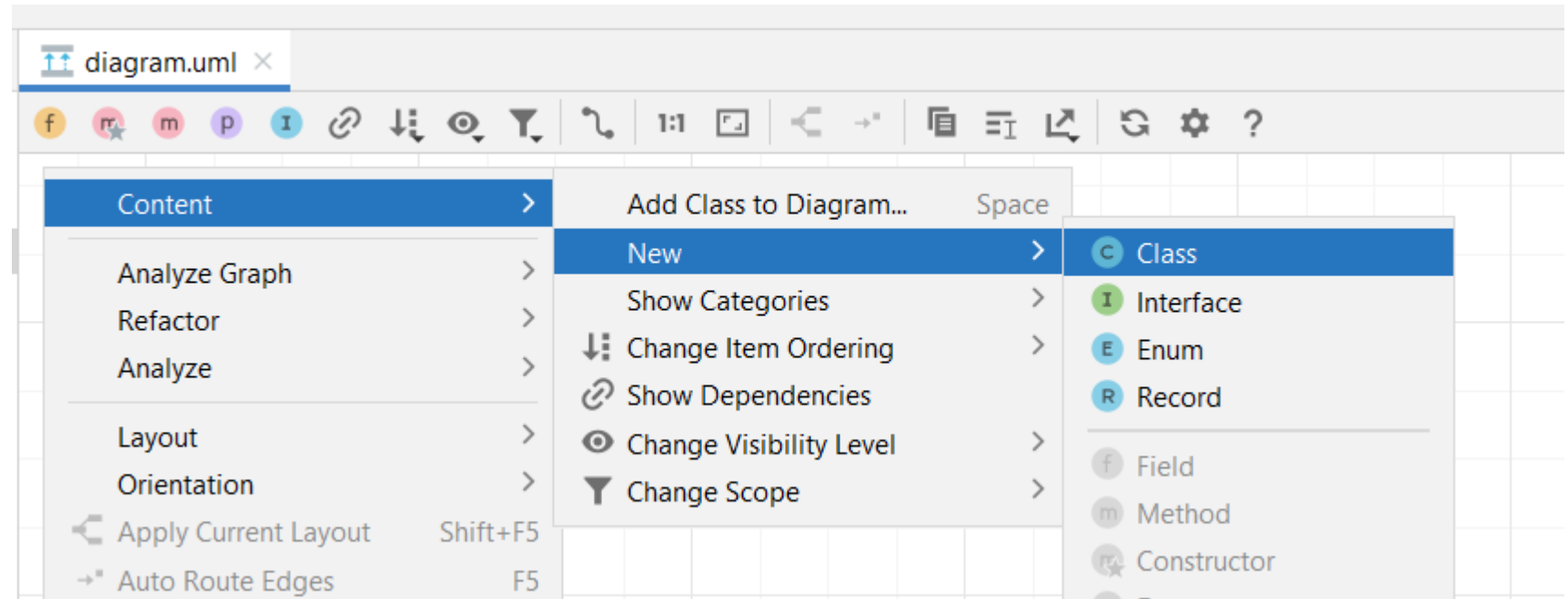
# UML in IntelliJ



Met naam mens.uml

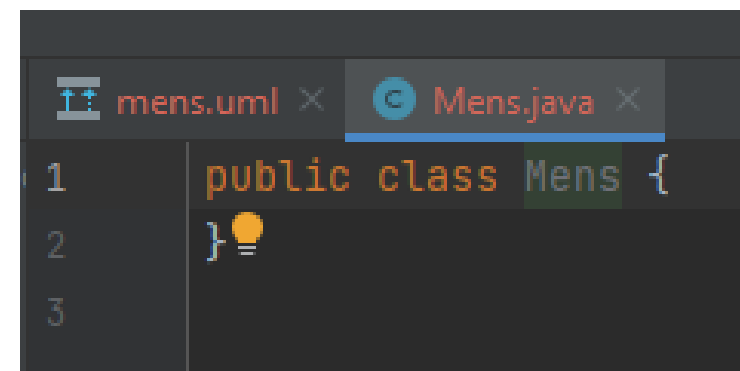
# UML in IntelliJ

- Voeg een klasse Mens toe

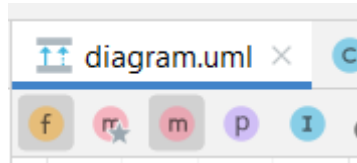


# UML in IntelliJ

- Automatisch ook code-file toegevoegd

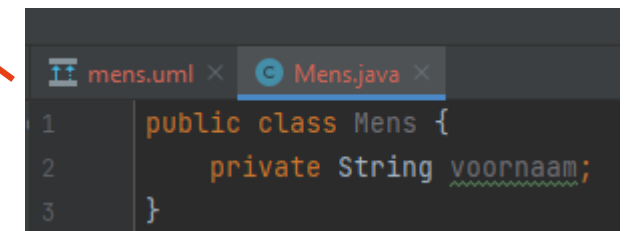
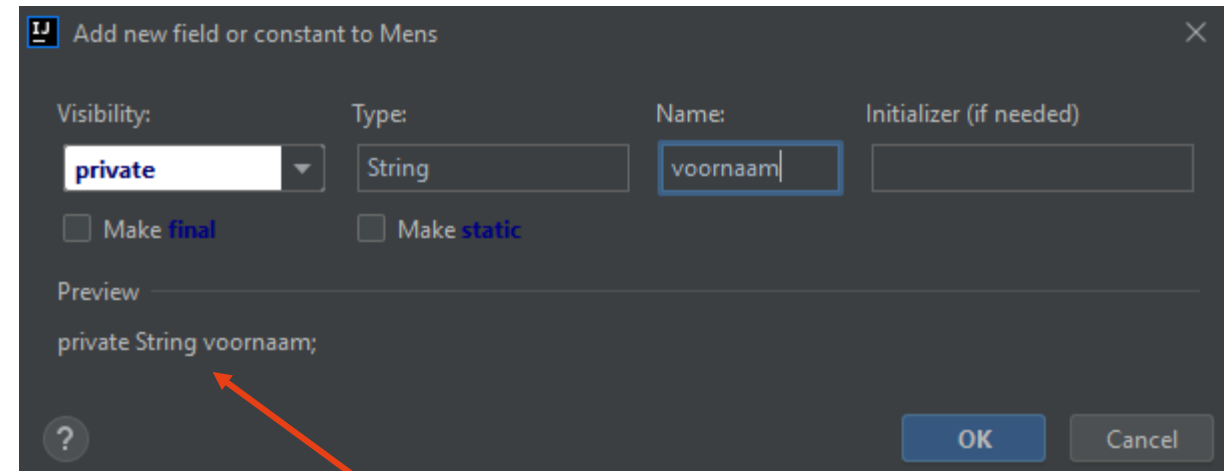
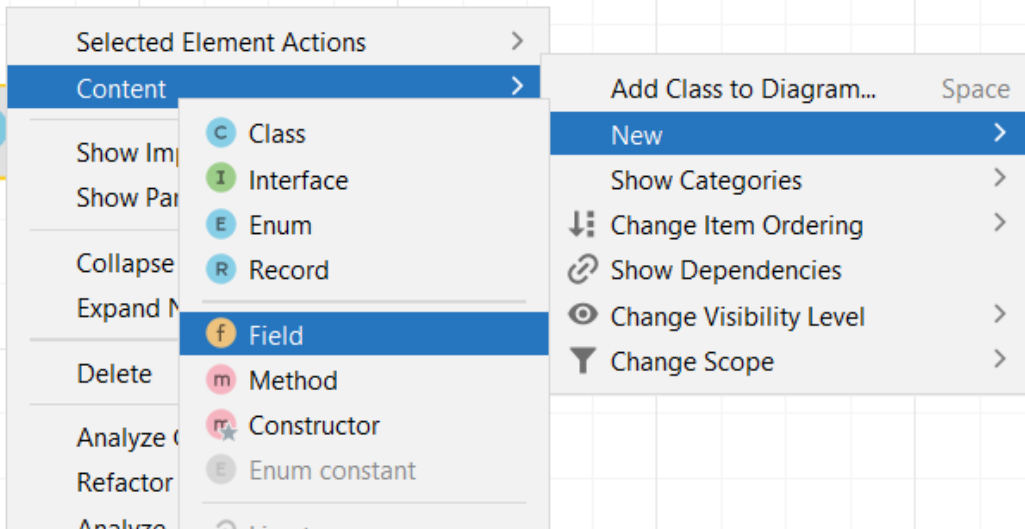


# UML in IntelliJ



Toon fields en methods

- Voeg properties (fields) / methoden / constructors toe



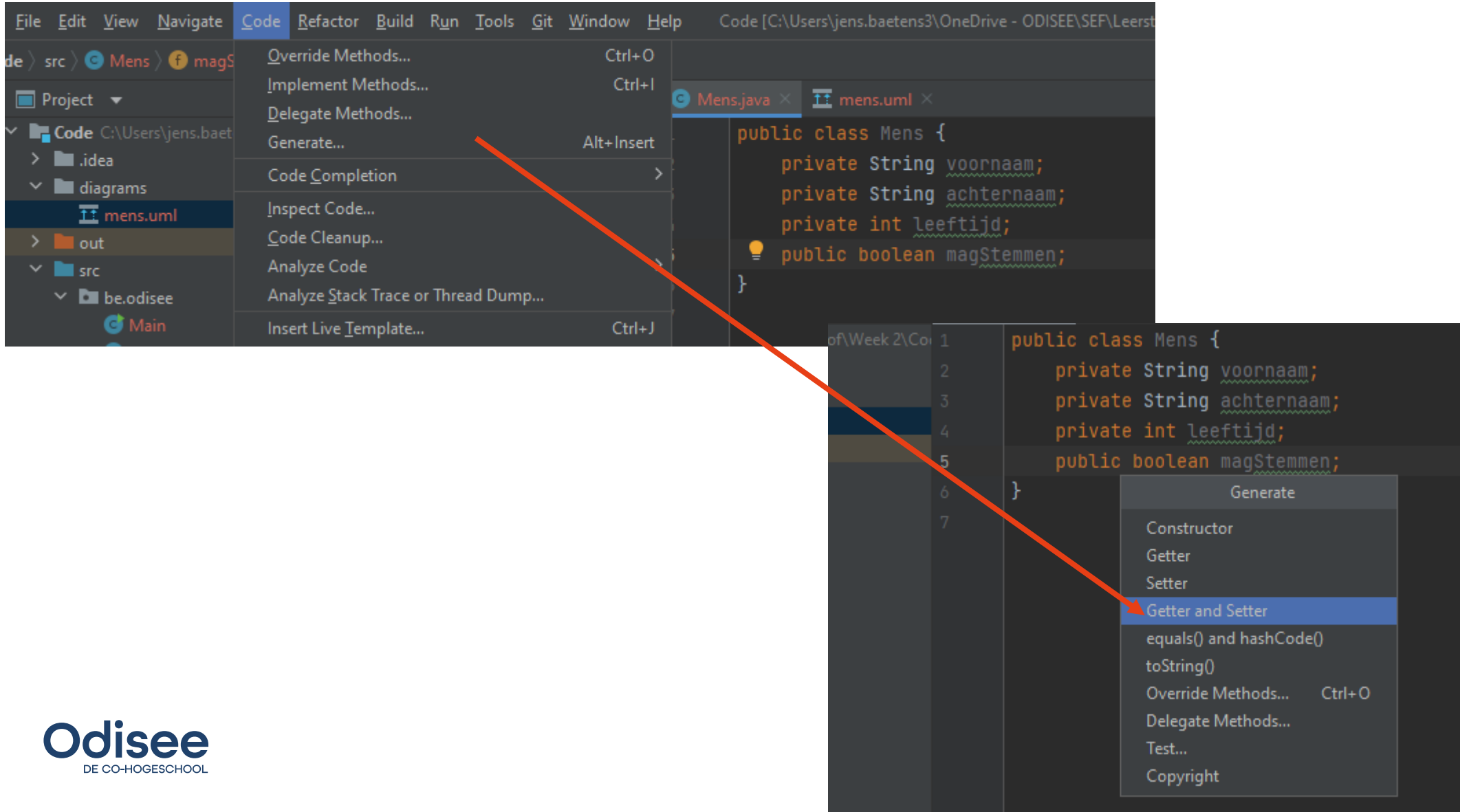




## UML in IntelliJ

- ▣ Oefening: Voeg de volgende properties toe
  - Achternaam: string – private
  - Leeftijd: int – private
  - MagStemmen: bool - public

## Voeg getter en setters toe voor de private properties



The screenshot shows the IntelliJ IDEA interface with the 'Code' menu open. The 'Generate' option is selected, which has opened a submenu. In this submenu, the 'Getter and Setter' option is highlighted. An orange arrow points from the 'Generate' button in the code editor to the 'Getter and Setter' option in the submenu.

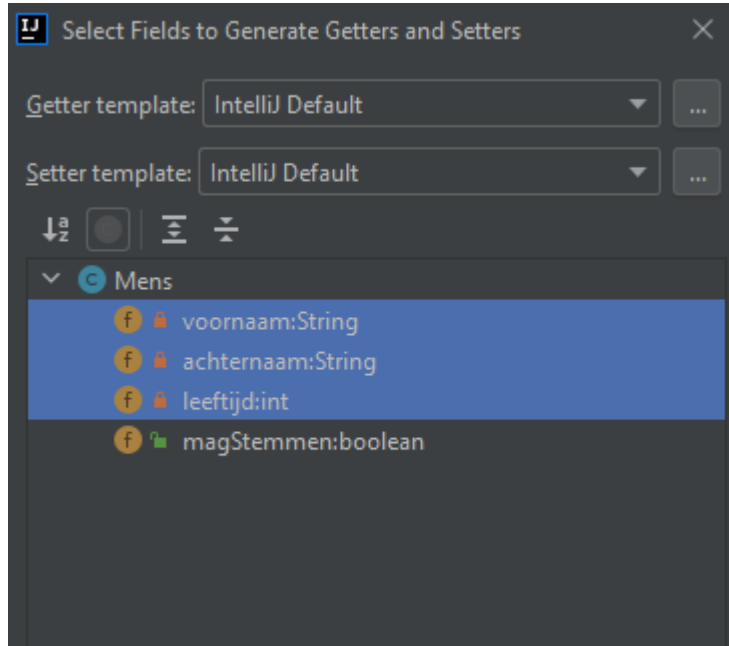
The code editor shows the following Java code for the 'Mens' class:

```
public class Mens {  
    private String voornaam;  
    private String achternaam;  
    private int leeftijd;  
    public boolean magStemmen;  
}
```

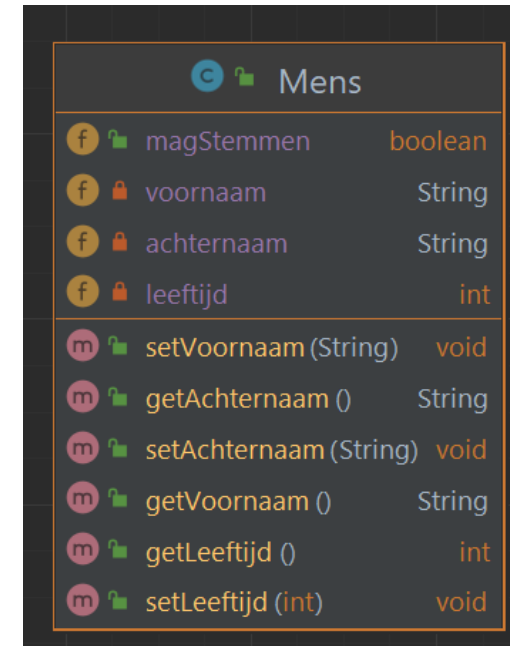
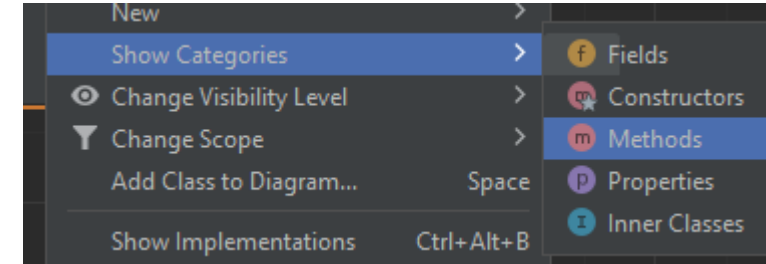
The submenu options are:

- Generate
- Constructor
- Getter
- Setter
- Getter and Setter
- equals() and hashCode()
- toString()
- Override Methods... Ctrl+O
- Delegate Methods...
- Test...
- Copyright

## Voeg getter en setters toe voor de private properties



```
public String getVoornaam() {  
    return voornaam;  
}  
  
public void setVoornaam(String voornaam) {  
    this.voornaam = voornaam;  
}  
  
public String getAchternaam() {  
    return achternaam;  
}  
  
public void setAchternaam(String achternaam) {  
    this.achternaam = achternaam;  
}
```



- Waarom moet de public property dit niet hebben?

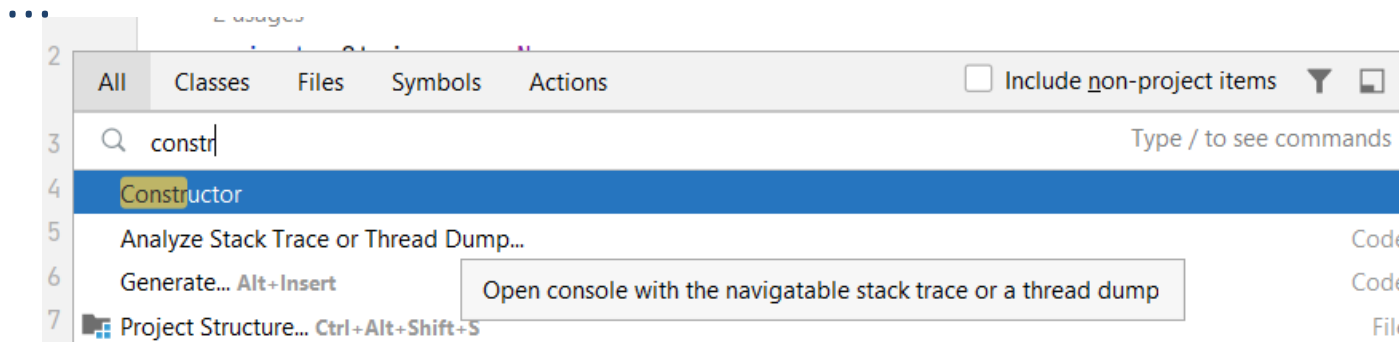


## Oefening: Genereer een constructor

- ▣ Welke properties moeten gekozen worden bij het genereren van de constructor?

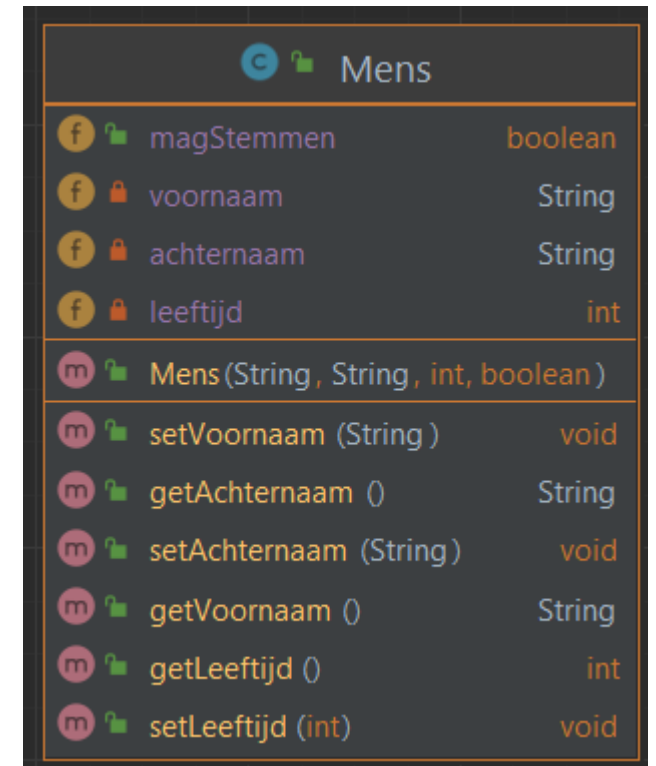
## Oplossing: Genereer een constructor

- Gebruik de shortcut shift + shift voor het opzoeken van commands, code,



## Resultaat

```
public Mens(String voornaam, String achternaam, int leeftijd, boolean magStemmen) {  
    this.voornaam = voornaam;  
    this.achternaam = achternaam;  
    this.leeftijd = leeftijd;  
    this.magStemmen = magStemmen;  
}
```



## Oefening: Voeg methode lach toe

- ▣ Voeg via UML een methode lach toe
  - ▬ Return type is void
  - ▬ Geen parameters nodig
- ▣ Controleer in code of de methode is toegevoegd
- ▣ Implementeer de functie
  - ▬ Print naam van de gebruiker in console gevolgd door “Hahahahaha”

## Oplossing: Voeg de methode lach toe

Create New Method

Visibility: [ ] Modifier: [none] Return type: void Name: lach

Parameters Exceptions

Nothing to show

Signature Preview

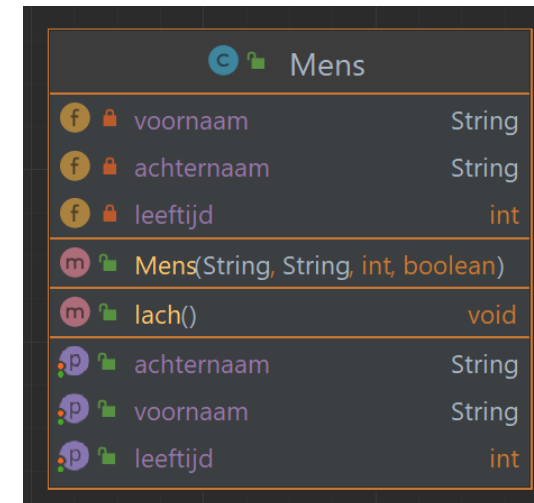
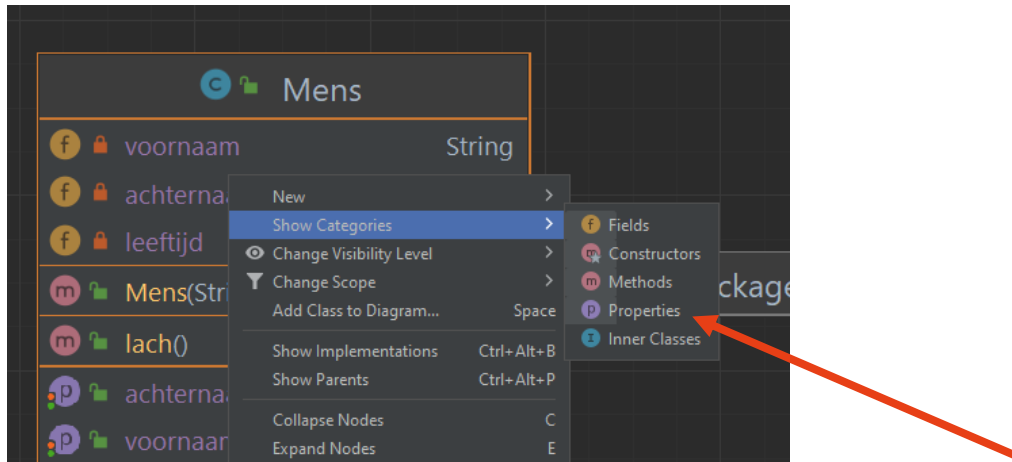
```
void lach()
```

Mens		
f	magStemmen	boolean
f	voornaam	String
f	achternaam	String
f	leeftijd	int
m	Mens (String, String, int, boolean)	
m	setVoornaam (String)	void
m	getAchternaam ()	String
m	lach ()	void
m	setAchternaam (String)	void
m	getVoornaam ()	String
m	getLeeftijd ()	int
m	setLeeftijd (int)	void

```
public void lach() {  
    System.out.println(voornaam + " Hahahahahahaha");  
}
```

## Overhead door getters en setters

- Getters en setters kunnen gegroepeerd worden door







# Test Driven Development

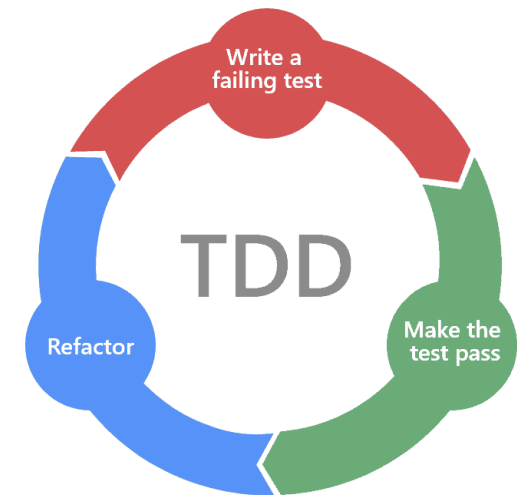
# Test driven development

- ▣ Software ontwikkelingsproces sterk gesteund op schrijven van testen

- ▣ 3 fasen

- ▢ Testing
- ▢ Coding
- ▢ Refactoring

- ▣ Kleine stappen met keer, volledig geteste code, ...



# Testing framework in Java

## ▣ Testing framework

- ▬ Zoeken van testen
- ▬ Automatisch uitvoeren van testen
- ▬ Bijhouden welke slagen en welke falen (gewenste output / exception)

## ▣ Junit

- ▬ Voor C# is er NUnit

## ▣ Voor unit testing

- ▬ Test “units” van code
- ▬ Unit = zo klein mogelijk deel (vaak 1 methode)

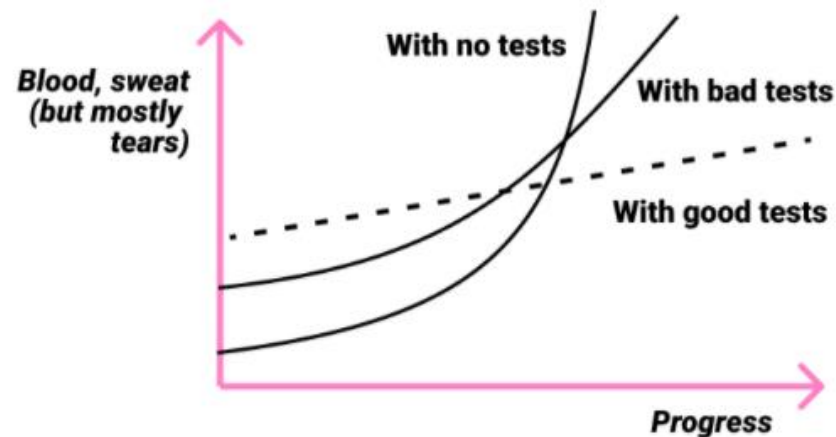
# Soorten testen

## ▣ Acceptatietesten

- Testen of je geschreven applicatie voldoet aan de eisen van de klant

## ▣ Regressietesten

- Testen of na wijzigingen niet-aangepaste code nog steeds werkt



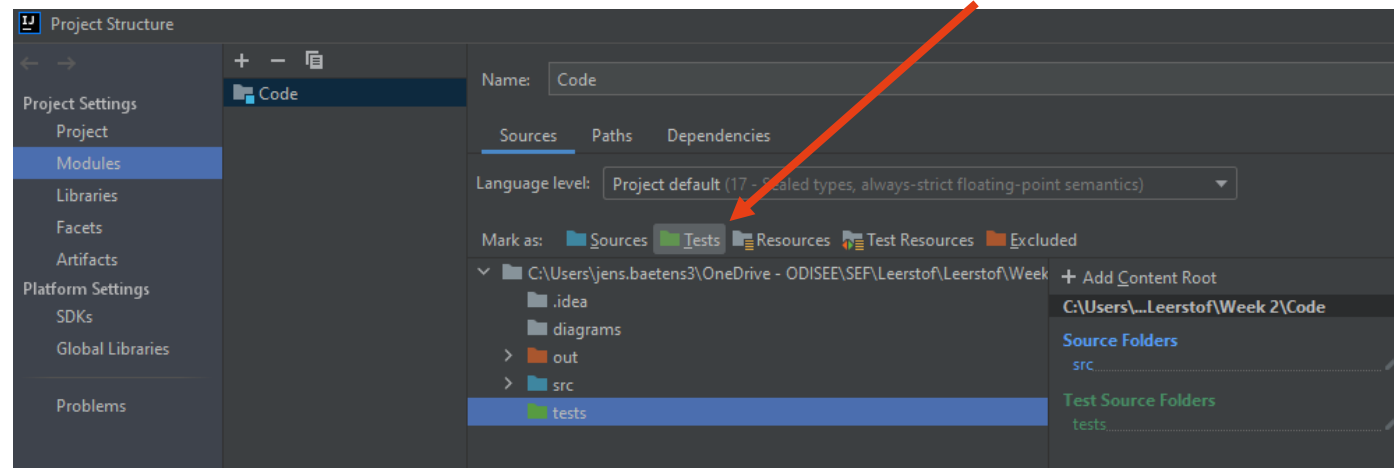
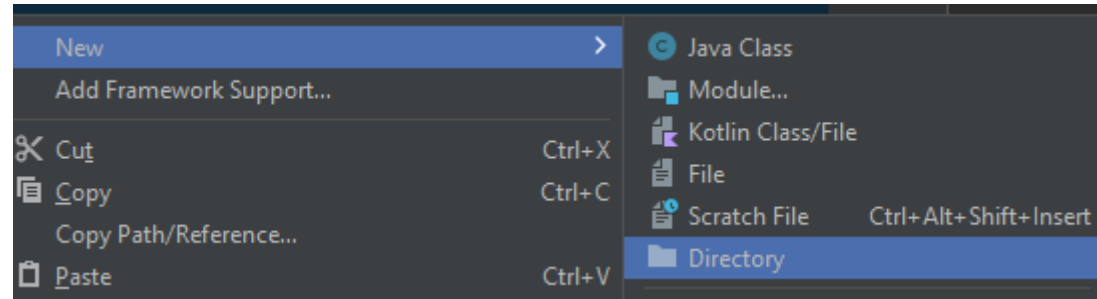
## Demo PetRock

- ▣ Maak eenvoudige klasse aan

```
public class PetRock {  
  
    private String name;  
  
    public PetRock(String name) {  
        this.name = name;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

# Demo PetRock

- Maak directory tests aan
- Maak deze directory de tests directory
  - File -> Project Structure -> Modules
  - Selecteer de tests directory
  - Klik bovenaan op Tests



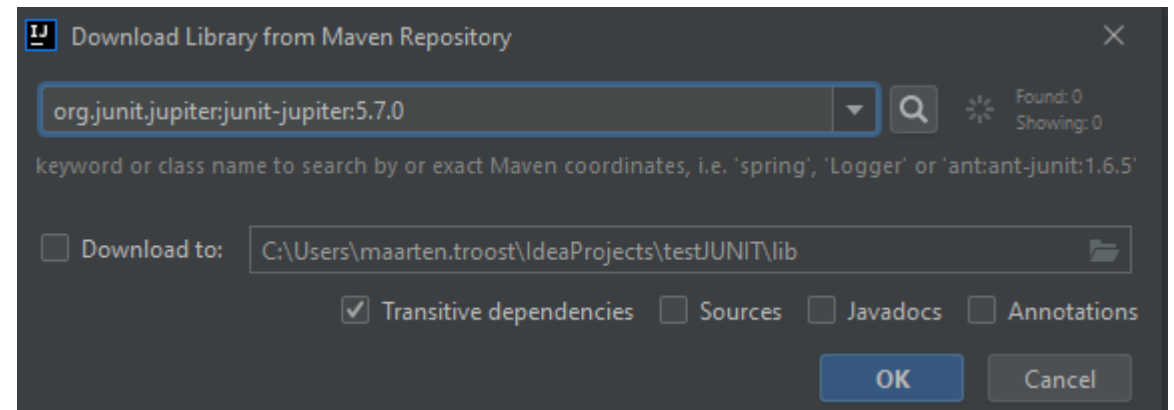
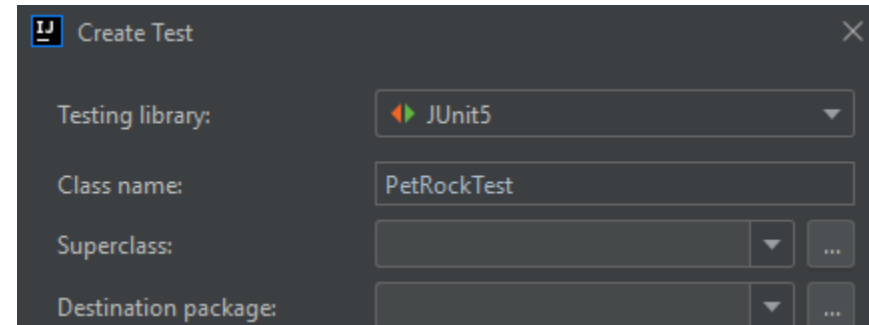
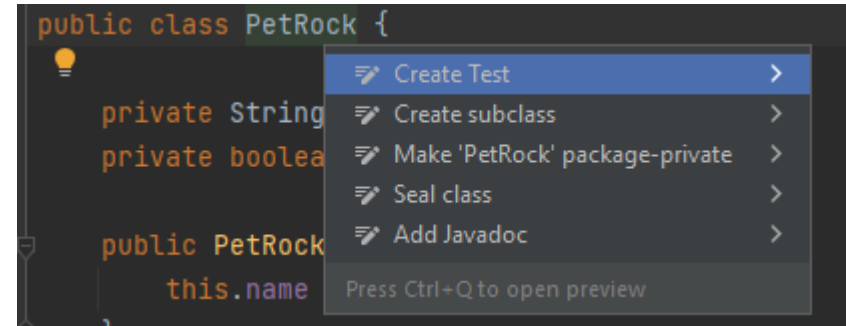
## Demo PetRock

### ■ Maak een TestKlasse aan

- Selecteer naam klasse
- Alt + Enter
- Create Test

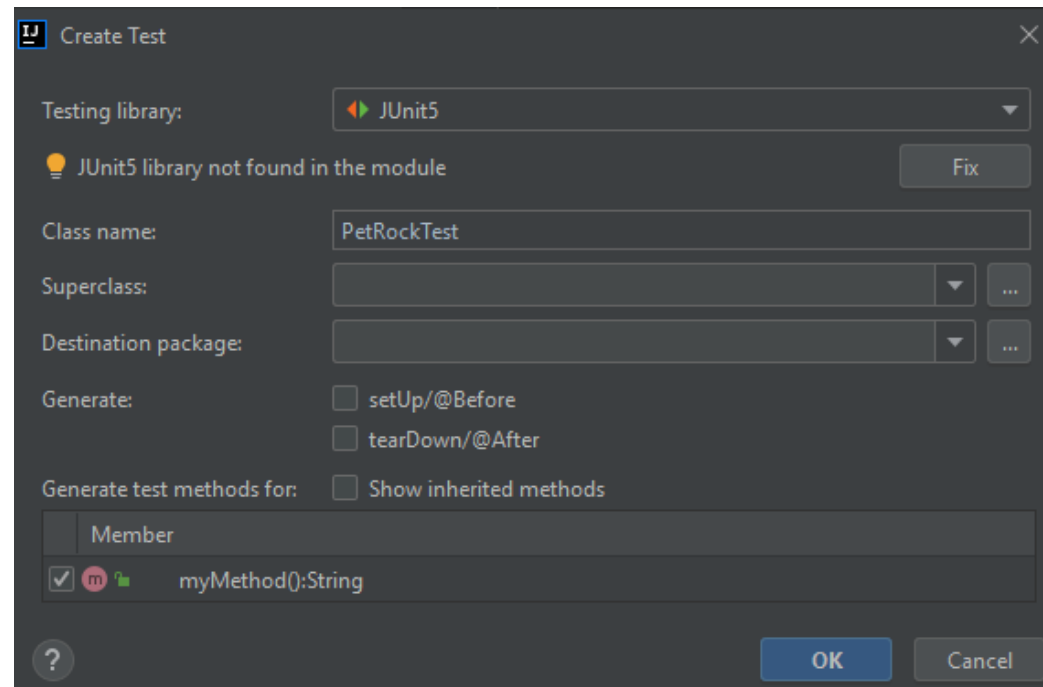
### ■ Toevoegen JUnit aan project

- Indien JUnit5 not found -> fix
- Verifieer org.junit.Jupiter
- OK



## Demo PetRock: installatie JUnit

- ▣ Kies een naam = naam te testen klasse + Test
- ▣ Testingframework Junit5
- ▣ Selecteer voor welke methoden er een test moet zijn





## Demo PetRock: TestKlasse

### ■ Annotatie @...Test

- ▬ Geeft aan dat de volgende methode een test is
- ▬ Indien hier een rode tag is, is er iets mis
  - Intellisense kan dit oplossen

```
import static org.junit.jupiter.api.Assertions.*;

class PetRockTest {

    @org.junit.jupiter.api.Test
    void getName() {
        PetRock rocky = new PetRock( name: "Rocky");

        assertEquals( expected: "Rocky", rocky.getName());
    }
}
```

- Testmethode maakt een object aan en test of de code doet wat we ervan verwachten
- Rechtsklik op de methode en kies run om de test uit te voeren



# Programmeren via test driven development

- ▣ Voeg een test toe dat controleert of de PetRock gelukkig is of niet
  - ▬ Bij default is deze niet gelukkig

```
@Test
void isHappy_AtStart_ReturnsFalse(){
    PetRock rocky = new PetRock( name: "Rocky");

    assertFalse(rocky.isHappy());
}
```

- ▣ Deze test faalt omdat de isHappy() methode niet bestaat

```
C:\Users\jens.baetens3\OneDrive - ODISEE\SEF\
java: cannot find symbol
  symbol:   method isHappy()
  location: variable rocky of type PetRock
```

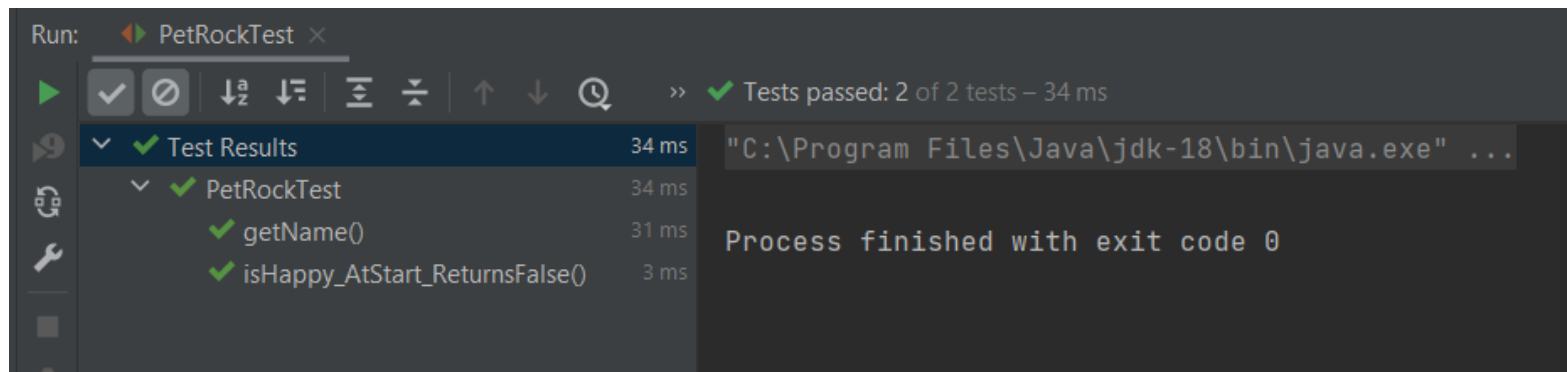
Test Results	34 ms	"C:\Program Files\Java\jdk-18\bin\jav
PetRockTest	34 ms	
✓ getName()	27 ms	
✗ isHappy_AtStart_ReturnsFalse()	7 ms	org.opentest4j.AssertionFailedError: Expected :false Actual :true <a href="#">&lt;Click to see difference&gt;</a>

# Programmeren via test driven development

- ▣ Nu dat test faalt, schrijf (minimale) code om test te doen slagen

```
public boolean isHappy(){  
    return false;  
}
```

- ▣ Alle testen slagen, schrijf opnieuw eerst test die faalt voor de nieuwe functionaliteit



## Programmeren via test driven development

- ▣ Volgende functionaliteit die we willen toevoegen is dat het object gelukkig is nadat ermee gespeeld is
  - ▢ Maak hierbij gebruik van een `.play()` methode
- ▣ Schrijf opnieuw eerst de test voor er gecodeerd wordt

# Programmeren via test driven development

```
@Test
void isHappy_AfterPlay_ReturnsTrue() {
    // Arrange or Given
    PetRock rocky = new PetRock( name: "Rocky");

    // Act or When
    rocky.play();

    // Assert or Then
    assertFalse(rocky.isHappy());
}
```

Waarom faalt deze test?



# Programmeren via test driven development

- ▣ Vervolledig de code om de test te doen slagen

```
public class PetRock {  
  
    private String name;  
    private boolean happy = false;  
  
    public PetRock(String name) {  
        this.name = name;  
    }  
  
    public String getName() { return name; }  
  
    public boolean isHappy(){  
        return false;  
    }  
  
    public void play(){  
        happy = true;  
    }  
}
```

## Best practices voor een goede test

- ▣ Leesbaarheid is heel belangrijk
  - ▬ Arrange – Act – Assert of Given – When – Then
- ▣ Act is maximaal 1 lijn, anders test je meerdere zaken
- ▣ In de assert moet je alle neveneffecten ook controleren en niet alleen de return waarde
  - ▬ Bvb als je een PetRock toevoegt aan je lijst van huisdieren

## Best practices – Arrange Act Assert

```
@Test
void isHappy_AfterPlay_ReturnsTrue() {
    // Arrange or Given
    PetRock rocky = new PetRock( name: "Rocky");

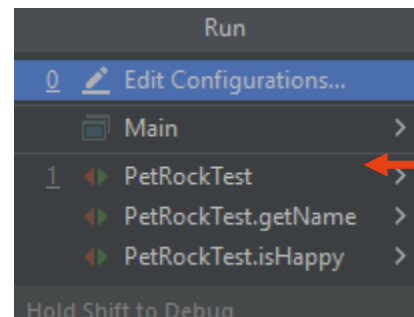
    // Act or When
    rocky.play();

    // Assert or Then
    assertFalse(rocky.isHappy());
}
```



## ▣ Belangrijke toetsen combinaties

- ▬ Alt + Enter om een testklasse aan te maken
  - Zorg ervoor via file -> project structure -> modules dat er een test directory is
- ▬ Alt + Shift + F10 om run config in te stellen
- ▬ Ctrl + Shift + F10 om geselecteerde config uit te voeren
  - Alle testen indien klasse geselecteerd
  - 1 test indien specifieke test geselecteerd



Alle testen uitvoeren  
in deze klasse

## Belangrijke annotaties

### ▣ @BeforeAll / @AfterAll

- Voor een gedeelde initialisatie van de testklasse
- Aanmaken data, opzetten connecties, downloaden informatie

### ▣ @BeforeEach / @AfterEach

- Voer initialisatie elke test opnieuw uit



# Documentatie

## Documentatie - Javadoc

- ▣ Platform voor genereren van documentatie voor Java applicatie
  - ▬ Zelfde stijl / manier voor alle applicaties
  - ▬ Gegengereerd op basis van documentatie in code
- ▣ Informatie over hoe de applicatie / klassen te gebruiken
- ▣ Voorbeeld: (google javadoc String)  
<https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/String.html>



## Manieren voor documentatie

- ▣ `// ...` voor single line
- ▣ `/* ... */` voor commentaar tussen code lijnen
  - Voor extra toelichting van de code maar niet in de documentatie
- ▣ `/** ... */`
  - Commentaar wordt opgevangen bij genereren van documentatie
  - Boven een methode / klasse
  - Voeg minstens een beschrijving van klasse / methode / parameters / return waarde toe

# Voorbeeld voor PetRock

```
/**
 * Deze klasse stelt een huisdier-steen voor met een naam en of de steen gelukkig is
 * @author Jens Baetens
 * @version 0.1
 */
public class PetRock {

    private String name;
    private boolean happy = false;

    /**
     * Constructor voor het aanmaken van het object met een naam, standaard geluk is false
     * @param name Naam van de steen
     */
    public PetRock(String name) {
        this.name = name;
    }

    /**
     * Getter voor de naam
     * @return de naam
     */
    public String getName() {
        return name;
    }
}
```

# Genereren documentatie

## ▣ Javadoc command line

```
C:\Users\jens.baetens3\OneDrive - ODISEE\SEF\Leerstof\Leerstof\Week 2\Code\src>javadoc -author -version -d ../documentation/ -subpackages .
```

Pad waar code staat

Commando

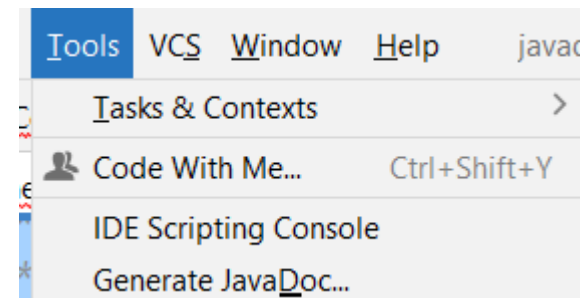
Waar moet html staan

In welke folder moet er gezocht worden

## ▣ IntelliJ IDEA:

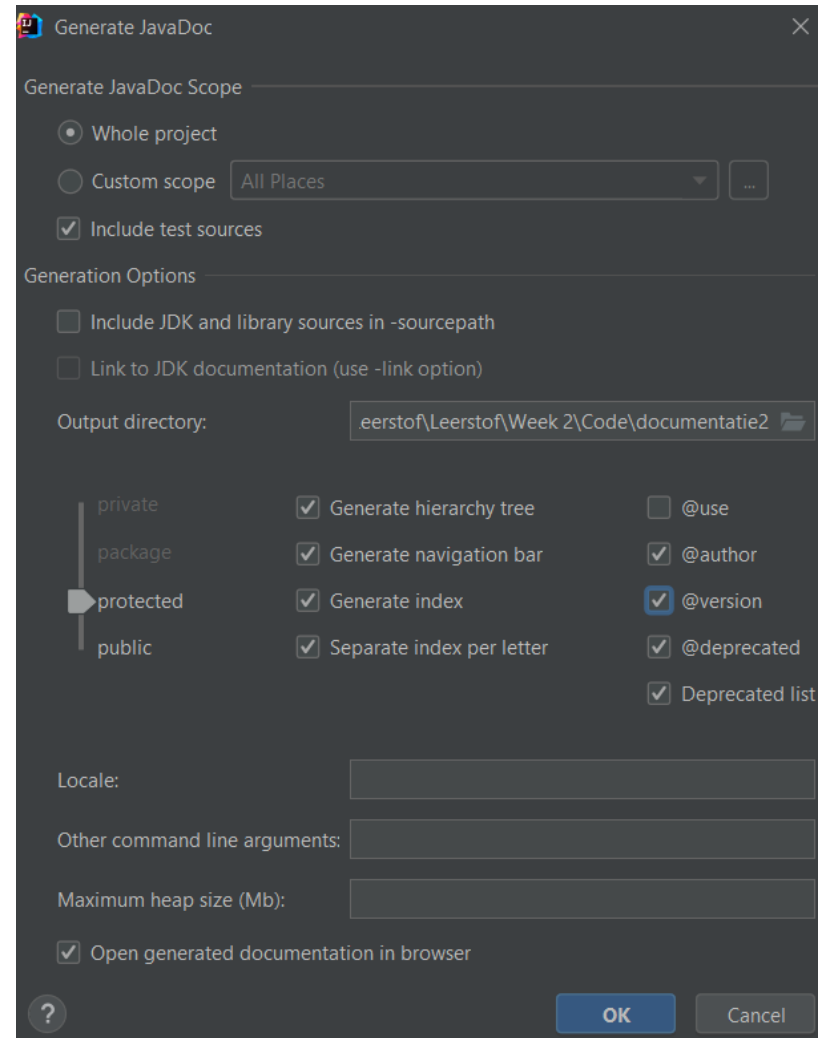
- Maak een directory javadoc aan in het project en genereer de documentatie via Tools > Generate JavaDoc.

## ▣ Zie ook Toledo > inhoud > Hoe java documentatie schrijven



# Genereren documentatie

- ▣ Javadoc command line
- ▣ Via IntelliJ





# Genereren documentatie

be.odisee

## Class PetRock

java.lang.Object  
be.odisee.PetRock

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

Deze klasse stelt een huisdier-steen voor met een naam en of de steen gelukkig is

**Version:**

0.1

**Author:**

Jens Baetens

### Constructor Summary

#### Constructors

##### Constructor and Description

**PetRock**(java.lang.String name)

Constructor voor het aanmaken van het object met een naam, standaard geluk is false

## Huiswerk / Opdracht





## Toledo leerobjecten

- ▣ Onder cursusdocumenten -> Leerobjecten week 2
- ▣ Neem het door, beluister de filmpjes en maak nota's samenvatting voor in het leerverslag

## In te dienen oefening

- ▣ De opdracht staat op github classroom en kan bereikt worden via de link
  - <https://classroom.github.com/a/jJZeQtP3>
- ▣ Dien de code in door het te pushen naar github
  - eventueel via IntelliJ git integratie
  - of externe git applicatie zoals github desktop
- ▣ Vergeet ook het leerverslag niet toe te voegen als pdf in de git repository