

JAVA PROGRAMMIEREN

Les 4 - Methodes

INHOUD

- Polymorfisme
- Abstracte klassen
- Huiswerkopdrachten
- Scope
- Access Modifiers
- Keywords
- Static
- Overloading

POLYMORFISME

*Wanneer je een superklasse definieert,
kan elke subklasse vervangen worden
door zijn supertype*

DIT GA JE ZOMETEEN SNAPPEN

STEL JE VOOR, JE HEBT DEZE CODE

```
1 public class Person
```

```
1 public class FamilyDoctor extends Person
```

```
1 Person person = new Person();
```

```
2 FamilyDoctor familyDoctor = new FamilyDoctor();
```

MAAR FAMILYDOCTOR EXTENDS PERSON

```
1 Person familyDoctor = new FamilyDoctor();
```

```
1 List<Person> persons = new ArrayList<>();  
2 persons.add(familyDoctor);
```

PRAKTIJKVOORBEELDJE

- Constructors beide klassen
- Wanneer welke methodes?
- Oplossing hiervoor?

OPDRACHT NAKIJKEN
POLYMORFISME LES 01

ABSTRACTE KLASSEN

ABSTRACTE KLASSEN

- Abstracte klassen kunnen niet geïnstantieerd worden
- Kunnen abstracte methodes bevatten
- Mag gewone methodes bevatten!

```
1 public abstract class Person
```

ABSTRACTE METHODES

- Hebben geen body
- Moeten geïmplementeerd worden in concrete subklassen.
- Gebruik `@Override` annotatie

ABSTRACTE METHODE - CODE

```
1 void work();  
2 String encryptPassword(String password);
```

```
1 @Override  
2 public void work() {  
3     System.out.println("Ja");  
4 }  
5 @Override  
6 String encryptPassword(String password) {  
7     return password + "encrypted";  
8 }
```

OPDRACHT BESPREKEN

- Overervig les 02
- Overervig opdracht 01

SCOPE

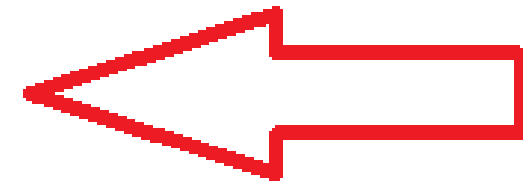
*Met scope bedoelen we de
toegankelijkheid en de
levensvatbaarheid van een variabele.*

VIER SCOPES

- Klasse scope
- Methode scope
- Loop scope
- Bracket scope

KLASSE SCOPE

```
public class Student {  
    private String firstName;  
    private String lastName;  
  
    public Student() {  
          
    }  
}
```



Aanpasbaar en
aanroepbaar vanuit
de gehele klasse

METHODE SCOPE

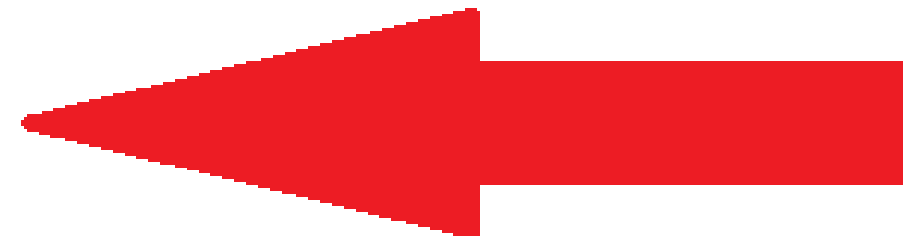
```
public void makeSound() {  
    int decibel = 12;  
    decibel = 100;  
}
```



Aanroepbaar binnen de
methode

```
public void snore() {  
    decibel = 30;  
}
```

Buiten methode niet
bereikbaar




Cannot resolve symbol 'decibel'

LOOP SCOPE (1)

```
public void printAllCourseNames() {  
    for (int i = 0; i < courses.size(); i++) {  
        System.out.println(  
            i + courses.get(i).getName()  
        );  
    }  
    System.out.println(i);  
}
```

Binnen de lus aanroepbaar en aanpasbaar

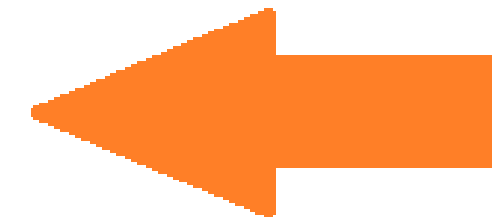
Variabele bestaat niet meer buiten de lus



Cannot resolve symbol 'i'

LOOP SCOPE (2.1)

```
public void printAllCourseCodes() {  
    for (Course course : courses) {  
        int amountOfCourses = 0;  
        System.out.println(  
            amountOfCourses  
            + course.getCourseCode()  
        );  
        amountOfCourses++;  
    }  
    System.out.println(amountOfCourses);  
}
```



LOOP SCOPE (2.2)

```
public void printAllCourseCodes() {  
    for (Course course : courses) {  
        int amountOfCourses = 0;  
        System.out.println(  
            amountOfCourses  
            + course.getCourseCode()  
        );  
        amountOfCourses++;  
    }  
    System.out.println(amountOfCourses);  
}
```

output

0a

0b

0c



BRACKET SCOPE

Gebruik deze gewoon niet.

```
1 {  
2   // Gebruik mij niet  
3 }
```

ACCESS MODIFIERS

THE FOUR

- private
- package private
- protected
- public

PUBLIC & PRIVATE

- Public: beschikbaar voor alle Java-code
- Private: beschikbaar binnen de klasse

PACKAGE PRIVATE

```
public class Soldier {  
    ? String name;  
}  
public class General {  
    public void changeNameSoldier(Soldier s) {  
        s.name = "Henk";  
    }  
}
```

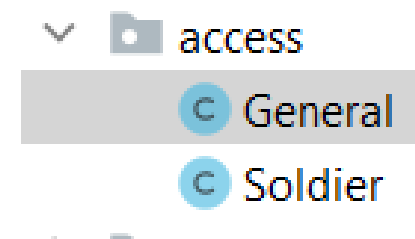
access
General
Soldier



Staan in dezelfde package

PACKAGE PRIVATE

```
public class Soldier {  
    ? String name;  
}  
public class General {  
    public void changeNameSoldier(Soldier s) {  
        s.name = "Henk";  
    }  
}
```



Staan in dezelfde package

Super handig voor encapsulatie

PROTECTED

- Toegang vanuit dezelfde package en de subklasse
- Subklasse mogen dus ook in een andere package staan!
- Super() werkt met protected.
- Beter dan public, want?

```
1 public abstract class Phone {
2     protected List<String> callTechnologies;
3     public Phone() {
4         this.callTechnologies = new ArrayList<>();
5         callTechnologies.add("2g");
6     }
7     protected abstract void call();
8 }
```

```
1 public class SmartPhone extends Phone {
2     public SmartPhone() {
3         super();
4         this.callTechnologies.add("3g");
5     }
6     @Override
7     protected void call() {
8         System.out.println("I call");
9     }
10 }
```

JA?

Even kijken in IntelliJ

- Wat als we het private maken?
- En public?
- En package private?
- En de klasse verplaatsen?

Overweeg je public access-modifiers op protected te zetten in de super-klasse

KEYWORDS

- Meer dan vijftig
- Meeste heb je al gezien
- Final!

FINAL!

Te gebruiken op drie niveau's

- Klasse-niveau
- Methode-niveau
- Instantievariabele-niveau (instance variable)

FINAL - KLASSE NIVEAU

- Voorkomt dat klasse extend kan worden.
- Dus de klasse kan niet als superklasse fungeren
- Wanneer je dit wel probeert, volgende foutmelding:
"Cannot inherit from final 'package.classname'".

FINAL - METHODE NIVEAU

- Voorkomt dat een methode overriden kan worden.
- Zeer handig als je een zeer strict contract wilt opstellen
- De vraag is of je dat wilt. Maakt je code minder uitbreidbaar!
- Geeft de volgende foutmelding:

`""methodName()' cannot override 'methodName()' in 'packages.ClassName'; overridden method is final"`

FINAL - INSTANCE VARIABLE

Wil je dat een waarde niet meer aangepast kan worden na toekenning. Dan gebruik je final

- Moet of direct een waarde gegeven worden
- of via (alle) constructor(s)

Final - Instance variable (2)

```
1 public class Car {  
2     private final int amountOfWheels = 4;  
3 }
```

Final - Instance variable (3)

```
1 public class Car {  
2     private final int amountOfWheels;  
3  
4     public Car(int amountOfWheels) {  
5         this.amountOfWheels = amountOfWheels;  
6     }  
7 }
```

Final - Instance variable (4)

```
1 public class Car {  
2     private final int amountOfWheels;  
3     private String engineType;  
4  
5     public Car(int amountOfWheels) {  
6         this.amountOfWheels = amountOfWheels;  
7     }  
8  
9     public Car(int amountOfWheels, String engineType) {  
10        this.amountOfWheels = amountOfWheels;  
11        this.engineType = engineType;  
12    }  
13 }
```

Final - Instance variable (5)

```
1 public class Car {  
2     private final int amountOfWheels;  
3     private String engineType;  
4  
5     public Car(String engineType) {  
6         this.engineType = engineType;  
7         this.amountOfWheels = 4;  
8     }  
9 }
```

TE GEBRUIKEN VOOR

- Waardes die uiteindelijk niet meer aangepast mogen worden
- Afdwingen dat instance variables een waarde krijgen

ERRORS VEROOORZAAKT DOOR FINAL

- Cannot inherit from final 'packages.ClassName'
- 'methodName()' cannot override 'methodName()' in 'packages.ClassName'; overridden method is final
- Cannot assign a value to final variable 'Variabele-Naam'
- Variable 'Variabele-Naam' might not have been initialized
- Illegal combination of modifiers: 'final' and 'abstract'

STATIC

- Static methods
- Static instance variables

STATIC METHODS

Is een methode die

- je kunt aanroepen zonder een klasse te instantiëren.
- Staat het keywordje *static* in.
- Werkt voor de rest hetzelfde als een normale methode
- **Maar** kan geen non-static methodes en variabelen aanroepen

CODEVOORBEELD

```
public class Utilities {  
    public static String greetUser(String name) {  
        return "Hello " + name;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Utilities.greetUser(name: "Nick");  
    }  
}
```

Geen new, geen instantiatie. Gebruik
klassenaam+methodenaam. Methodenaam cursief.

STATIC INSTANCE VARIABLE

- Gelijk voor alle instanties van de klasse
- Met *final* klasse-brede constante van te maken.

VOORBEELD IN INTELLIJ

OVERLOADING

*Zelfde methode naam, andere
parameters*

Volgorde & type is hier belangrijk.

VOORBEELDJE

```
1 public class Person {  
2     public void eat(String food) {  
3         System.out.println("I eat " + food);  
4     }  
5     public void eat(String food, int amountKg) {  
6         System.out.println("I eat " + amountKg + " of " + food);  
7     }  
8 }
```

VOORBEELDJE DAT NIET MAG

```
1 public void eat(String food) {  
2     System.out.println("I eat " + food);  
3 }  
4 public void eat(String ingredient) {  
5     System.out.println("I eat ingredient: " + ingredient);  
6 }
```

`eat(String)` is already defined in `packages.Person`

H & H

- **Geen** fysiek koffiemoment morgen
- Koffiemoment van 0900-1200 morgen
- Maak overige lesopdrachten
- Bedenk wat je nog niet helemaal snapt en geef dat aan
- Overige lees-huis-werk komt uiterlijk morgenavond online.