

Software development 6A&D

Enkele afspraken

Inhoud

- Vakafpraken
- Wat moet je studeren?
 - ✓ De slides
 - ✓ Oefeningen
 - ✓ Eventuele extra meegegeven weblinks / video's
 - ✓ **Notities** tijdens de les
 - ✓ Afkortingen + terminologie: geen schrijffouten
- Oefening baart kunst!



Software development 6A&D

**Inleiding tot object-georiënteerd
programmeren in JAVA. (OOP)**

Inhoud

- OOP = Object Oriented Programming
- Klasse en objecten
- Methodes
- Information hiding / encapsulation

OOP = Object Oriented Programming

- **Vroeger :**

enkel aandacht voor wat een programma moet « doen »
(procedureel programmeren) en te weinig voor de
gegevens

- **Nu :**

zowel de « gegevens » als de « acties erop» zijn verpakt in
OBJECTEN

Evolutie naar OOP

Today We Will Learn:

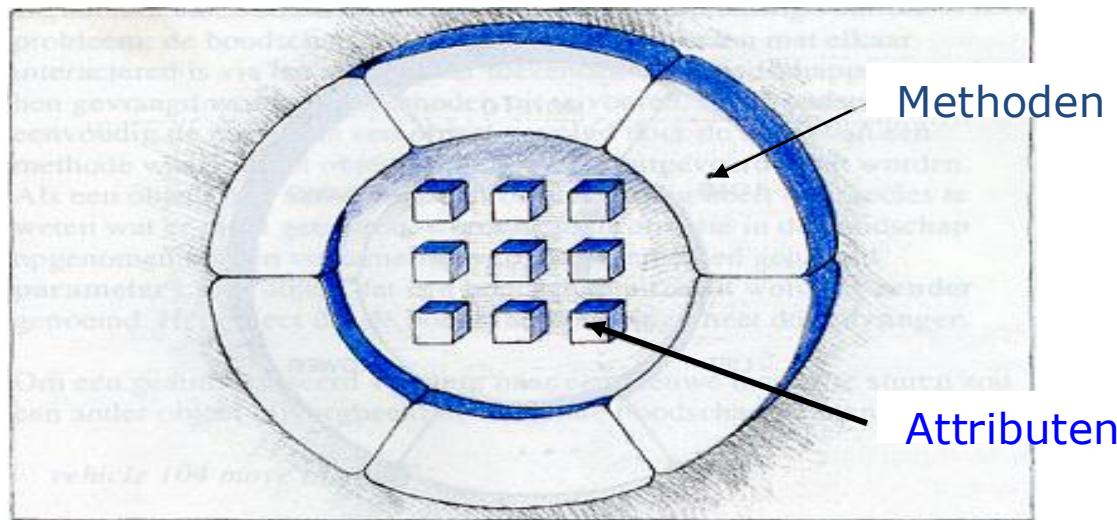
3. Understand OOP with Real Life Examples



<https://www.youtube.com/watch?v=1wJfIUcVWIA&t>

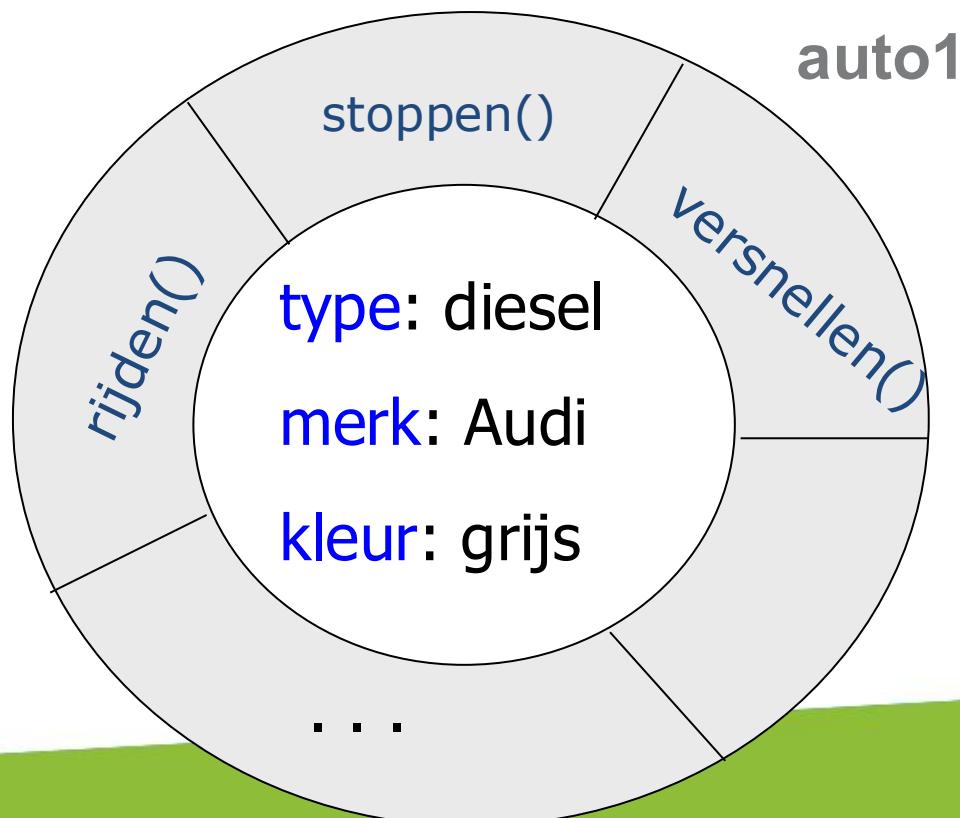
Klasse en objecten

- Alles rondom ons is een **object**
- Elk object heeft kenmerken (**attributen** = eigenschappen)
- Elk object kan acties ondernemen (**methoden**)

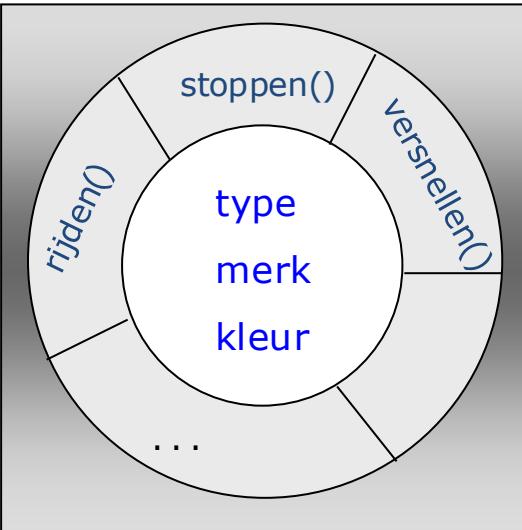


Klasse en objecten

- Object = een auto ergens rondom ons genaamd "auto1"
- Attributen = wat zijn de eigenschappen "auto1"?
- Methoden = wat kan "auto1" doen?

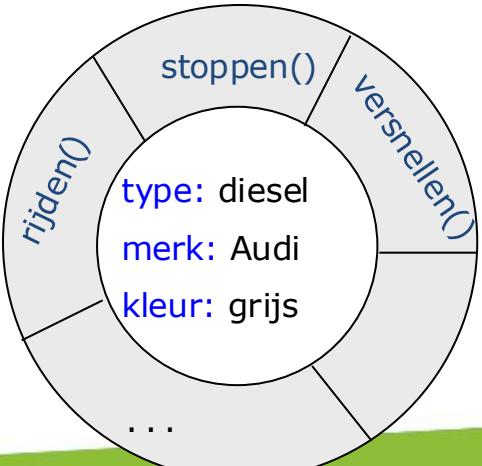


Klasse en objecten

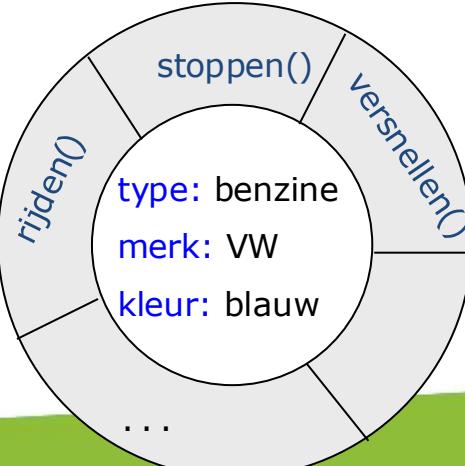


Klasse Auto

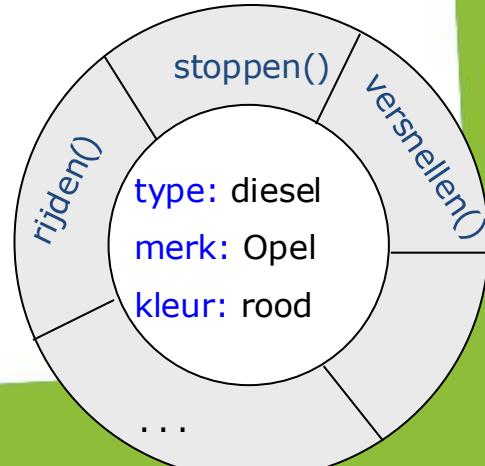
object auto1



object auto2



object auto3

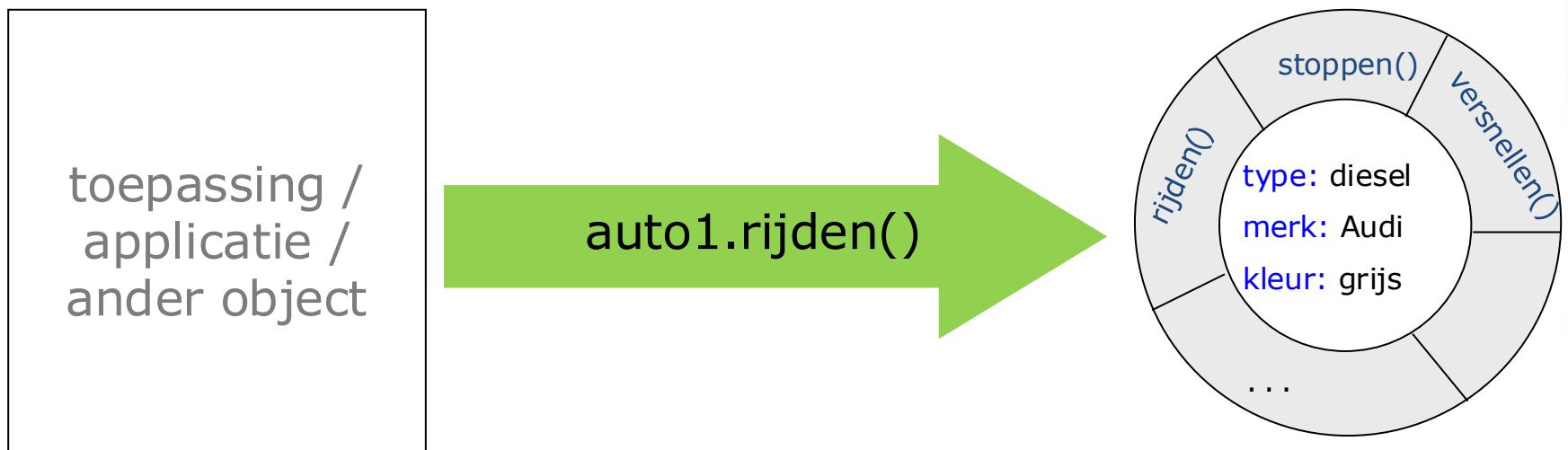


Klasse en objecten

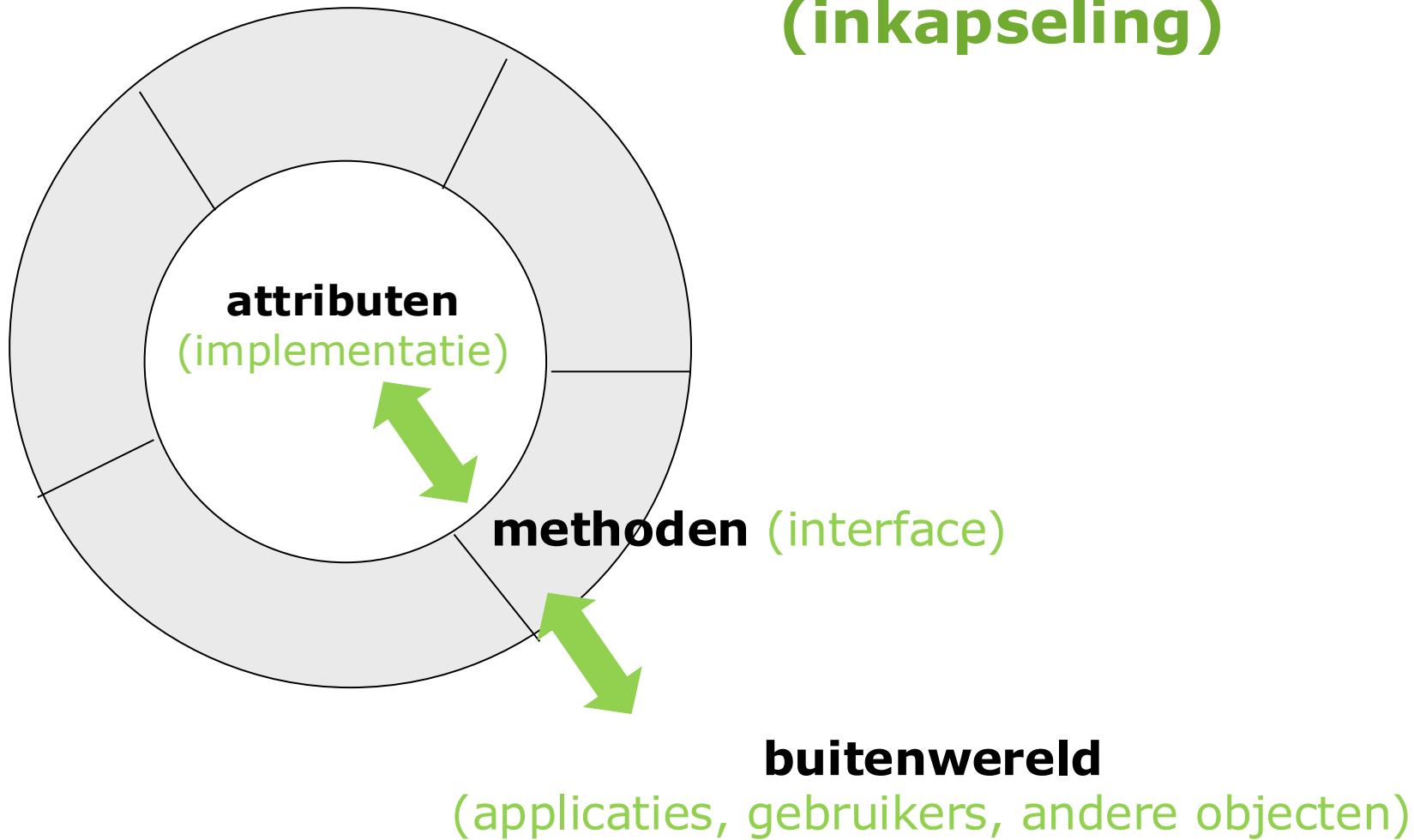
- **Klasse**: beschrijving van attributen (velden, gegevens) en gedrag (methoden, functies) van gelijksoortige objecten.
- **Object** van de klasse: een instantie in overeenstemming met beschrijving van klasse.
- Je kan van een klasse zoveel **instanties** maken (instantiëren) als je zelf wilt: auto1, auto2, auto3...
- Elke instantie van een klasse beschikt over dezelfde attributen (aantal en soort) en dezelfde methoden.

Methodes

- Acties ondernemen.
- Het aanroepen van een methode noemt men ook vaak "*het sturen van een bericht (message)*"



Information hiding / encapsulation (inkapseling)



Software development

6A&D

**Zelf een klasse maken in JAVA.
(Basisstructuur)**

Zelf een klasse maken in Java

Basisstructuur:

```
public class NaamKlasse  
{  
    //inhoud van de klasse  
}
```

Klasse bevat:

- ✓ **attributen:** gegevens die het object moet gebruiken
- ✓ **methoden:** implementatie van gedrag van object

Voorbeeldklasse: Toetsenbord

- We werken in Java een klasse Toetsenbord uit.
- **Attributen:**
 - ✓ serienummer
 - ✓ merk
 - ✓ is het draadloos of niet
 - ✓ prijs
- **Mogelijke methodes:**
 - ✓ Om de waarde van een attribuut aan te passen
 - ✓ Om de waarde van een attribuut op te vragen
 - ✓ ...



Software development 6A&D

**Zelf een klasse maken in JAVA.
(Attributen & methoden)**

Voorbeeldklasse: attributen

```
public class Toetsenbord {  
    //velden = attributen  
    private int serienummer;  
    private String merk;  
    private boolean draadloos;  
    private double prijs;
```

- Attributen zijn steeds **private**
- Methodes zijn meestal **public**: het moet immers mogelijk zijn boodschappen te sturen naar een object
- Klasse Toetsenbord moet opgeslagen worden in een bestand met de naam **Toetsenbord.java**

Voorbeeldklasse: methoden

- In Java moet je voor elke methode specifiëren:
 - ✓ *access modifier*
 - ✓ **returntype**
 - ✓ **naam van de methode**
 - ✓ **parameter(s)**
- Bijvoorbeeld

```
public boolean vergelijk (int getal) {...}  
public int grootste (String tekst, String woord) {...}  
public void verander (char letter) {...}  
private double bereken () {...}  
public void druk () {...}  
public void veranderLayout (Layout layout) {...}
```

Voorbeeldklasse: algemene opbouw

```
public class Toetsenbord {
```

Tijdelijke versie

//Attributen

```
private int serienummer;  
private String merk;  
private boolean draadloos;  
private double prijs;  
private Layout layout;
```

//Methoden

```
public double getPrijs() { ... }  
public void setMerk(String merk) { ... }  
public Layout getLayout(){ ... }
```

```
}
```

Software development 6A&D

**Zelf een klasse maken in JAVA.
(Methoden: setters & getters)**

Voorbeeldklasse: setters & getters

Een klasse wordt voorzien van "getters" om attribuutwaarden op te vragen, "setters" om attribuutwaarden te wijzigen en andere methodes die nodig zijn.

Getter = accessor methode

- ✓ methode waarmee je waarde van attribuut opvraagt
- ✓ verandert toestand van object niet
- ✓ naam begint met '**get**' (of '**is**' in het geval van een boolean) gevuld door **naam** van attribuut waarvan de waarde wordt opgevraagd
- ✓ bvb. **getSerienummer()**, **getPrijs()**, **isDraadloos()**

Setter = mutator methode

- ✓ methode die de toestand van een object verandert
- ✓ naam begint met '**set**' gevuld door **naam** van attribuut
- ✓ Bvb. **setDraadloos(true)**, **setMerk("Philips")**

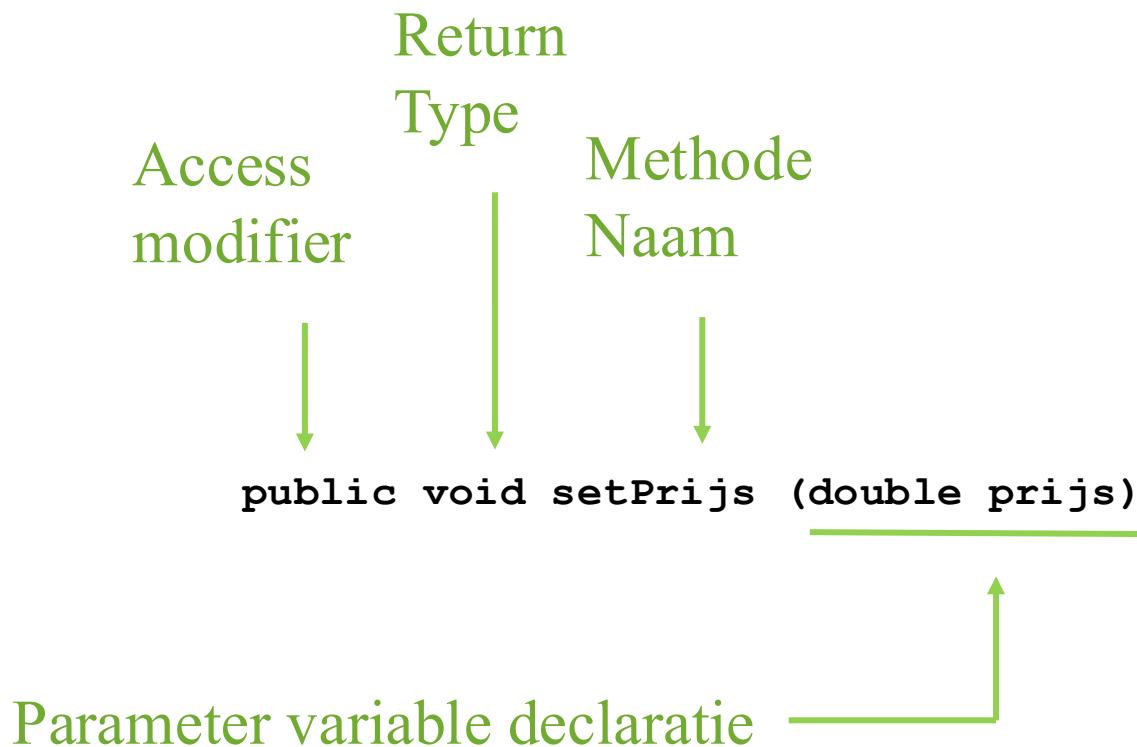
Voorbeeldklasse: Toetsenbord

```
//getters
public int getSerienummer() {
    return serienummer;
}
public String getMerk() {
    return merk;
}
public boolean isDraadloos() {
    return draadloos;
}
public double getPrijs() {
    return prijs;
}
public Layout getLayout() {
    return layout;
}
```

Voorbeeldklasse: Toetsenbord

```
//setters
public void setSerienummer(int sn) {
    serienummer = sn;
}
public void setMerk(String merkje) {
    merk = merkje;
}
public void setDraadloos(boolean draad) {
    draadloos = draad;
}
public void setPrijs(double prijsje) {
    prijs = prijsje;
}
public void setLayout(Layout lay) {
    layout = lay;
}
```

Header van de setPrijs methode



Gebruik van this

```
public void setPrijs (double prijsje) {  
    prijs = prijsje;  
}
```



- Je bent verplicht een andere naam te verzinnen voor de parameter om het onderscheid te maken met het attribuut
- Doe je dit niet? Dan negeert de compiler de parameter.
- Het is beter/gemakkelijker gebruik te maken van **this**
- **this** is een referentie naar het object zelf.

Je bent dan niet verplicht voor elke parameter een nieuwe naam te kiezen.

Gebruik van this

```
//setters
public void setSerienummer(int serienummer) {
    this.serienummer = serienummer;
}
public void setMerk(String merk) {
    this.merk = merk;
}
public void setDraadloos(boolean draadloos) {
    this.draadloos = draadloos;
}
public void setPrijs(double prijs) {
    this.prijs = prijs;
}
public void setLayout(Layout layout) {
    this.layout = layout;
}
```

Voorbeeldklasse: algemene opbouw

```
public class Toetsenbord {  
  
    //Attributen  
    private int serienummer;  
    private String merk;  
    private boolean draadloos;  
    private double prijs;  
    private Layout layout;  
  
    //Methoden  
  
    //Setters en getters  
    public int getSerienummer() { return serienummer; }  
    public void setSerienummer(int serienummer) { this.serienummer = serienummer; }  
    public String getMerk() { return merk; }  
    public void setMerk(String merk) { this.merk = merk; }  
    public boolean isDraadloos() { return draadloos; }  
    public void setDraadloos(boolean draadloos) { this.draadloos = draadloos; }  
    public double getPrijs() { return prijs; }  
    public void setPrijs(double prijs) { this.prijs = prijs; }  
    public Layout getLayout() { return layout; }  
    public void setLayout(Layout layout) { this.layout = layout; }  
}
```

Tijdelijke versie

Software development

6A&D

Zelf een klasse maken in JAVA.
(Methoden)

Voorbeeldklasse: Toetsenbord

```
//Methoden  
public double getPrijsDollar(double omreken)  
{  
    double dollars; //gebruik lokale variabele  
    dollars = prijs * omreken;  
    return dollars;  
}  
  
public double getPrijsDollar(double omreken)  
{  
    return prijs * omreken;  
}
```

Opgelet!

Beide methoden kunnen niet samen geïmplementeerd worden.

Attribuutvariabele ≠ lokale variabele

- **Attribuutvariabele**

- ✓ = een EIGENSCHAP van een object
- ✓ wordt **buiten** een methode (maar in een klasse) gedeclareerd
- ✓ is gekend in alle methoden van die klasse
- ✓ wordt automatisch geïnitialiseerd (zie uitleg constructor)

- **Lokale variabele**

- ✓ = variabele die (tijdelijk) nodig is binnen een methode
- ✓ wordt **in** een methode gedeclareerd
- ✓ wordt NIET door private voorafgegaan
- ✓ is alleen in deze methode gekend
- ✓ kan gebruikt worden vanaf de plaats van declaratie tot aan de sluitaccoade van het blok waarin de declaratie staat.
- ✓ wordt niet automatisch geïnitialiseerd

Voorbeeldklasse: algemene opbouw

Tijdelijke versie

```
public class Toetsenbord {  
  
    //Attributen  
    private int serienummer;  
    private String merk;  
    private boolean draadloos;  
    private double prijs;  
    private Layout layout;  
  
    //Methoden  
    public double getPrijsDollar(double omreken) {...}  
  
    //Setters en getters  
    public int getSerienummer() { return serienummer; }  
    public void setSerienummer(int serienummer) { this.serienummer = serienummer; }  
    public String getMerk() { return merk; }  
    public void setMerk(String merk) { this.merk = merk; }  
    public boolean isDraadloos() { return draadloos; }  
}
```

Software development

6A&D

**Zelf een klasse maken in JAVA.
(Constructors en object aanmaken)**

Constructors

- Klassen hebben speciale methoden “constructors” genoemd
- Een constructor is een methode die **automatisch** opgeroepen wordt als een object wordt gecreëerd
- Constructors doen alles wat te maken heeft met het **initialiseren** van het object zoals het initialiseren van de attributen

Constructors

- Constructors hebben speciale eigenschappen die hen verschillend maakt van normale methoden:
 - ✓ Constructors hebben **dezelfde naam** als de klasse
 - ✓ Constructors hebben **geen return type** (ook niet void)
 - ✓ Constructors zijn meestal **public**

Constructor voor de Rechthoek Klasse

Een constructor om een object van de klasse Rechthoek te initialiseren zou er zo kunnen uitzien:

Opgelet: er is geen return type in een constructor

```
public Rechthoek(double lengte, double breedte) {  
    this.lengte = lengte;  
    this.breedte = breedte;  
}
```

Een object creëren van de klasse Rechthoek

- Een object kan gedeclareerd worden zonder te initialiseren:
Rechthoek doos;
- Dit statement creëert **geen** Rechthoek object; het is dus een referentievariabele waar je nog niets mee kan doen.
- Een referentievariabele die je wil gebruiken zonder dat deze geïnitialiseerd is, zal een compiler error geven:

```
C:\Users\Nick\IdeaProjects\untitled\src\Main.java:9:9  
java: variable doos might not have been initialized
```

Een object creëren van de klasse Rechthoek

- Hoe initialiseren? Hoe een object maken?

Rechthoek doos;

doos = **new** Rechthoek(**7.0, 14,5**);

of

Rechthoek doos = **new** Rechthoek(**7.0, 14,5**);

- Doos refereert nu naar een Rechthoek object met lengte 7.0 en breedte 14.5

Software development

6A&D

**Zelf een klasse maken in JAVA.
(Soorten constructors)**

De Constructor

- We kunnen drie soorten constructors onderscheiden:
 - ✓ De default constructor
 - ✓ De No-Arg constructor
 - ✓ Eigen definieerde constructor met argumenten

De Default Constructor

- Wanneer een object wordt gecreëerd, wordt **altijd** zijn constructor opgeroepen.
- Als je zelf geen constructor schrijft, zal Java een maken op het moment van compilatie. Deze constructor is de **default constructor**
- Alle attribuutvariabelen krijgen dan afhankelijk van hun type een default waarde:

| | |
|------------------|-------|
| ✓ int | 0 |
| ✓ double | 0.0 |
| ✓ boolean | false |
| ✓ char | " |
| ✓ String | ? |
| ✓ andere Klassen | null |
- Voor lokale variabelen moet je zelf een initialisatie voorzien!

De Default Constructor

- De default constructor is een constructor **zonder parameters** die gebruikt wordt om een object te initialiseren in een 'default' configuratie
- Java zal enkel een default constructor aanmaken als je geen enkele constructor in de klasse geschreven hebt
 - ✓ Zie voorbeeld: Toetsenbord.java
- Wanneer je wel een constructor hebt geschreven zal Java geen constructor meer toevoegen
 - ✓ Zie voorbeeld: Rechthoek.java met constructor

De “No-Arg Constructor”

- Een constructor zonder parameters noemt men een **no-arg constructor**
- De default constructor (die Java voorziet) is een no-arg constructor
- We kunnen ook onze eigen no-arg constructor schrijven (die de default constructor zal vervangen)

```
public Rechthoek() {  
    this.lengte = 1.0;  
    this.breedte = 2.0;  
}
```

Voorbeeldklasse: Toetsenbord

- Voorbeeld van een eigen no-arg constructor (die de default constructor vervangt)

```
public Toetsenbord() {  
    serienummer=987450;  
    merk = "?";  
    draadloos = true;  
}
```

Code zal deze constructor oproepen

- Hoe maak je nu een Toetsenbord-object?
(bijvoorbeeld in je Main.java)

```
Toetsenbord klavier;  
klavier = new Toetsenbord();
```

Naam van het object dat gemaakt wordt

Voorbeeldklasse: Toetsenbord

- Het object klavier heeft de volgende attribuutwaarden:
 - ✓ serienummer = 987450
 - ✓ merk = "?"
 - ✓ draadloos = true
 - ✓ prijs = 0.0
 - ✓ layout = null

Waarde die default wordt toegekend

```
↳ klavier = {Toetsenbord@795}
    f serienummer = 987450
    f merk = "?"
    f draadloos = true
    f prijs = 0.0
    f layout = null
```

Een eigen gedefinieerde constructor

- Een constructor met minstens één parameter.
- Indien een attribuut geen waarde wordt toegekend zal deze de default waarde krijgen.

```
public Toetsenbord(int serienummer, String merk) {  
    this.serienummer = serienummer;  
    this.merk = merk;  
    this.prijs = 100.0;  
}
```

```
Toetsenbord klavier = new Toetsenbord(123456, "Corsair");
```

```
  ↘  └─ klavier = {Toetsenbord@795}  
      └─ f  serienummer = 123456  
  >   └─ f  merk = "Corsair"  
      └─ f  draadloos = false  
      └─ f  prijs = 100.0  
      └─ f  layout = null
```

Voorbeeldklasse: algemene opbouw

Finale versie

```
public class Toetsenbord {  
  
    //Attributen  
    private int serienummer;  
    private String merk;  
    private boolean draadloos;  
    private double prijs;  
    private Layout layout;  
  
    //Constructors (0, één of meerdere)  
    public Toetsenbord() {...}  
    public Toetsenbord(int serienummer, String merk) {...}  
    public Toetsenbord(int serienummer, String merk, boolean draadloos, double prijs, Layout layout) {...}  
  
    //Methoden  
    public double getPrijsDollar(double omreken) {...}  
  
    //Setters en getters  
    public int getSerienummer() { return serienummer; }  
    public void setSerienummer(int serienummer) { this.serienummer = serienummer; }  
    public String getMerk() { return merk; }  
}
```

Software development

6A&D

**Zelf een klasse maken in JAVA.
(String & `toString()`-methode)**

De String Class Constructor

- Eén van de **String** klasse constructors aanvaardt een parameter van het type **String**.
- Bijvoorbeeld:

```
String naam = new String("Albert Einstein");
```

- De variabele **naam** refereert nu naar een **String** object met de waarde "Albert Einstein"

'new String' is redundant



[Replace with argument](#) Alt+Shift+Enter [More actions...](#) Alt+Enter

- Omdat we echter **String** objecten zo vaak gebruiken, is Java zo gemaakt dat we ook de volgende verkorte notatie kunnen gebruiken:

```
String naam = "Albert Einstein";
```

toString()-methode

- De methode `toString()` is handig om op een snelle manier inzicht te krijgen in de toestand van een object.
- De methode levert de waarde van het opgegeven object terug als String-waarde
- De methode is standaard voorzien in Java.



```
Object.java
237
238     public String toString() {
239         return getClass().getName() + "@" + Integer.toHexString(hashCode());
240     }
```

(*CTRL + linker muisklik om naar de desbetreffende klasse en methode te navigeren*)

toString()-methode

- De `toString()`-methode wordt opgeroepen wanneer je het object rechtstreeks aanspreekt of wanneer je deze expliciet aanroeft:

```
Toetsenbord klavier = new Toetsenbord(123456, "Corsair");
System.out.println("Het object");
System.out.println(klavier);
System.out.println("De methode");
System.out.println(klavier.toString());
```



```
"C:\Program Files\Java\jdk-14.0.1\bin\java.exe"
Het object
Toetsenbord@3feba861
De methode
Toetsenbord@3feba861
```

toString()-methode

- Aangeraden wordt op de standaard implementatie te overschrijven (**override**): het doel is dat de methode de inhoud van al de attributen van een object als **String** aflevert.

```
public class Toetsenbord {  
  
    @Override  
    public String toString() {  
        return "Toetsenbord{" +  
            "serienummer=" + serienummer +  
            ", merk=\"" + merk + '\"' +  
            ", draadloos=" + draadloos +  
            ", prijs=" + prijs +  
            ", layout=" + layout +  
            '}';  
    }  
}
```

toString()-methode

```
Toetsenbord klavier = new Toetsenbord(123456, "Corsair");
System.out.println("Het object");
System.out.println(klavier);
System.out.println("De methode");
System.out.println(klavier.toString());
```



```
"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" "-javaagent:D:\Program Files\JetBrains\IntelliJ IDEA 2020.3.3\lib\idea_rt.jar=5432,D:\Program Files\JetBrains\IntelliJ IDEA 2020.3.3\bin" -Dfile.encoding=UTF-8
Het object
Toetsenbord{serienummer=123456, merk='Corsair', draadloos=false, prijs=100.0, layout=null}
De methode
Toetsenbord{serienummer=123456, merk='Corsair', draadloos=false, prijs=100.0, layout=null}
```