

Software Development 6A&D

**De 'is-een'-relatie, ook wel overerving
of inheritance genoemd.**

Inleiding

- In het eerste semester zagen we een eerste soort van relaties in de OOP-wereld, de zogenaamde 'heeft een'-relatie, ook wel object als attribuut genoemd.
- Nu zullen we een tweede soort relatie zien, namelijk de 'is een'-relatie, ook wel overerving of inheritance genoemd.

Voorbeeld: Toetsenbord & Beeldscherm

- De klasse **Toetsenbord** met 4 attributen:

- ✓ serienummer
- ✓ merk
- ✓ prijs
- ✓ draadloos

- Elke computer heeft ook een **Beeldscherm** nodig, dus we kunnen daar ook een klasse voor ontwerpen. We voorzien de attributen

- ✓ serienummer
- ✓ merk
- ✓ prijs
- ✓ grootte

In beide klassen ontdekken we **gelijke attributen**, die trouwens ook zullen terugkomen bij andere computeronderdelen. Er zullen ook **gelijke methodes** zijn.

Voorbeeld: Toetsenbord & Beeldscherm

- Daarom:
 - ✓ we ontwerpen een klasse **Computeronderdeel** waarin we die gemeenschappelijke attributen en methodes plaatsen. Dit noemen we de *superklasse*
 - ✓ we maken daarna duidelijk dat **Toetsenbord** een speciaal geval is van **Computeronderdeel**. Toetsenbord noemen we een *subklasse*
 - ✓ ook **Beeldscherm** is een *subklasse*.
- Dit mechanisme heet *overerving* (*inheritance*)

Het UML-diagram

superklasse:
-> protected attribuut

Computeronderdeel

#serienummer:Integer
#merk:String
#prijs:BigDecimal

+Computeronderdeel()
+Computeronderdeel(merk:String)
+getSerienummer():Integer
+setSerienummer(serienummer:Integer):void
+getMerk():String
+setMerk(merk:String):void
+getPrijs():BigDecimal
+setPrijs(prijs:BigDecimal):void
+toString():String



Toetsenbord

-draadloos:boolean

+Toetsenbord()
+Toetsenbord(merk:String;prijs:BigDecimal)
+isDraadloos():boolean
+setDraadloos(draadloos:boolean):void
+toString():String

subklasse: extends in Java

Klasse Computeronderdeel

```
public class Computeronderdeel {  
    protected Integer serienummer;  
    protected String merk;  
    protected BigDecimal prijs;  
  
    public Computeronderdeel() {  
        serienummer = null; merk = ""; prijs = null;  
    }  
  
    public Computeronderdeel(String merk) {  
        serienummer = null; this.merk = merk; prijs = null;  
    }  
  
    public String toString() {  
        String tekst;  
  
        tekst = "Serienummer: " + serienummer + "\n";  
        tekst += "Merk: " + merk + "\n";  
        tekst += "Prijs: " + prijs + "\n";  
        return (tekst);  
    }  
  
    //Getters en setters  
}
```

Computeronderdeel – Toetsenbord

Computeronderdeel

Attribut(en):

- serienummer
- merk
- Prijs

Methode(n):

- getSerienummer
- getMerk
- getPrijs
- setSerienummer
- setMerk
- setPrijs
- getPrijsDollar
- toString

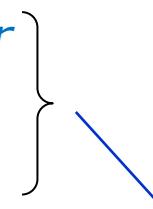
Toetsenbord

Attribut(en):

- *serienummer*
- *merk*
- *prijs*
- draadloos

Methode(n):

- *getSerienummer*
- *getMerk*
- *getPrijs*
- *setSerienummer*
- *setMerk*
- *setPrijs*
- *getPrijsDollar*
- *isDraadloos*
- *setDraadloos*
- eigen *toString*



Via
overerving



Klasse Toetsenbord

```
public class Toetsenbord extends Computeronderdeel {  
    private boolean draadloos;  
  
    public Toetsenbord() {  
        super();  
        draadloos = false;  
    }  
  
    public Toetsenbord(String merk, BigDecimal prijs) {  
        super(merk);  
        this.prijs = prijs; draadloos = false;  
    }  
  
    public String toString() {  
        String tekst;  
  
        tekst = super.toString();  
        tekst += "Draadloos: " + draadloos + "\n";  
        return (tekst);  
    }  
  
    //Getters en setters  
}
```

Toetsenbord instanties maken

```
Toetsenbord ducky = new Toetsenbord("Ducky", 139.99);
```

- via super(merk): wordt de tweede constructor van **Computeronderdeel** eerst opgeroepen waardoor **merk** de waarde "Ducky" krijgt
- via this.prijs=prijs: krijgt **prijs** de waarde 139,99.
!protected
- **serienummer** is een protected-variabele van **Computeronderdeel** en die is op null geïnitialiseerd
- **draadloos** is een private-variabele van **Toetsenbord** en die is standaard op false geïnitialiseerd

Samengevat

Principe van overerving:

- Uit een bestaande klasse kan je een andere nieuwe klasse maken
 - ✓ *oorspronkelijke klasse* = *superklasse*
 - ✓ *nieuwe klasse* = *subklasse*
- Subklasse beschikt automatisch over **dezelfde** attributen en methoden als superklasse **en** ook **eigen** attributen en methoden hebben

Samengevat

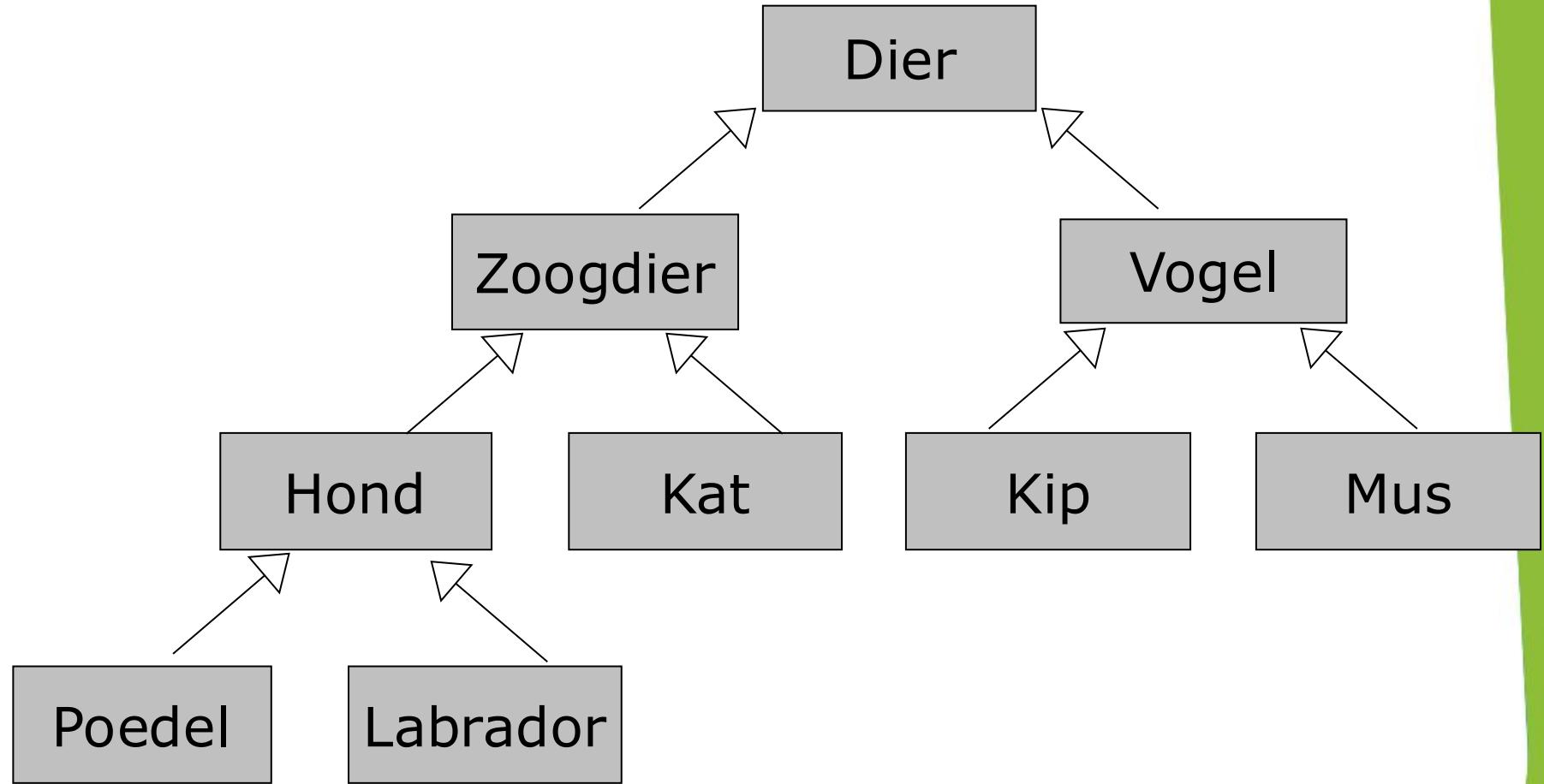
Principe van overerving:

- Via het keyword **super** verwijst je naar de superklasse. Dit kan voor een constructor **super(...)**, methode **super.methode()** of attribuut **super.attribuut**.
- In Java geeft **extends** de overervingsrelatie aan.
- Een superklasse kan verschillende subklassen hebben, maar elke subklasse maar 1 superklasse. Java heeft **enkelvoudige inheritance!**

Voordelen van overerving

- De bestaande superklasse blijft intact
- De subklasse kan zelf extra eigen attributen en methodes krijgen
- Voorkomen van geduplicateerde code
- Hergebruik van code
- Gemakkelijker onderhoud
- Uitbreidbaarheid

Een voorbeeld van een overervinghiërarchie



Software Development

6A&D

Overerving
& inkapseling (information hiding)

public, private en protected

1) delen als **public** gedefinieerd :

- ✓ toegankelijk voor objecten uit andere klassen
- ✓ public delen uit de klasse **Computeronderdeel** zijn toegankelijk voor de subklassen **Toetsenbord** en **Beeldscherm**
- ✓ methodes & constructors zijn meestal **public**

2) delen als **private** gedefinieerd :

- ✓ niet toegankelijk voor objecten uit andere klassen
- ✓ private delen uit de klasse **Computeronderdeel** zijn niet toegankelijk voor de subklassen **Toetsenbord** en **Beeldscherm**
- ✓ attributen zijn bijna altijd **private**

3) Om een attribuut toegankelijk te maken (zonder gebruik te maken van de setter) in een subklasse moet je in de superklasse private vervangen door protected.

Overzicht access modifiers

- **private:** klasselid is enkel toegankelijk vanuit instanties van deze klasse
- **public:** klasselid is van overal toegankelijk
- **protected:** klasselid is toegankelijk vanuit alle klassen binnen package en vanuit alle subklassen (eventueel buiten package)
- **niets vermelden** = package toegang, het klasselid is dan toegankelijk vanuit alle klassen binnen de package

Let dus op:

Vergeet het woord private niet, anders geef je package toegang!

Overzicht toegangsregels

```
package eerste;
```

```
class X {  
    private int a;  
    protected int c;  
    public int d;  
    int e;  
}
```

```
class P {  
    . . .  
}
```

```
package tweede;
```

```
class Y extends X {  
    . . .  
}
```

```
class Q {  
    . . .  
}
```

Overzicht toegangsregels: private

```
package eerste;
```

```
class X {  
    private int a;  
    protected int c;  
    public int d;  
    int e;  
}
```

```
class P {  
    . . .  
}
```

```
package tweede;
```

```
class Y extends X {  
    . . .  
}
```

```
class Q {  
    . . .  
}
```

Alleen toegankelijk in de klasse zelf

Overzicht toegangsregels: protected

```
package eerste;
```

```
class X {  
    private int a;  
    protected int c;  
    public int d;  
    int e;  
}
```

```
class P {  
    . . .  
}
```

```
package tweede;
```

```
class Y extends X {  
    . . .  
}
```

```
class Q {  
    . . .  
}
```

Overzicht toegangsregels: public

```
package eerste;
```

```
class X {  
    private int a;  
    protected int c;  
public int d;  
    int e;  
}
```

```
class P {  
    . . .  
}
```

```
package tweede;
```

```
class Y extends X {  
    . . .  
}
```

```
class Q {  
    . . .  
}
```

Overzicht toegangsregels: package

```
package eerste;
```

```
class X {  
    private int a;  
    protected int c;  
    public int d;  
int e;  
}
```

```
class P {  
    . . .  
}
```

```
package tweede;
```

```
class Y extends X {  
    . . .  
}
```

```
class Q {  
    . . .  
}
```