

Les 5 – Hergebruik van code

Compositie

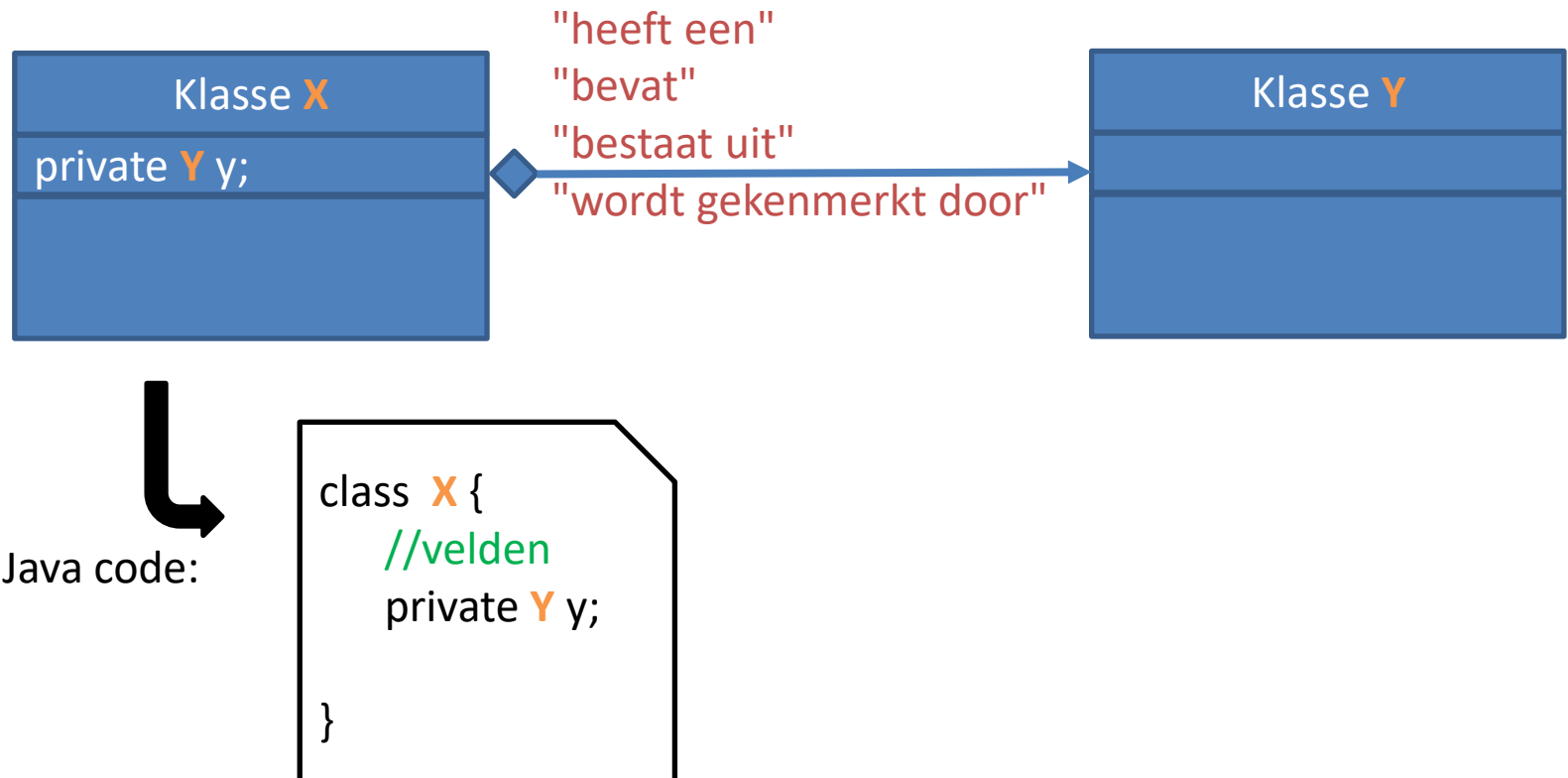
Overerving

Hergebruik van code



UML – Compositie relatie

"heeft een" relatie



Concreet voorbeeld

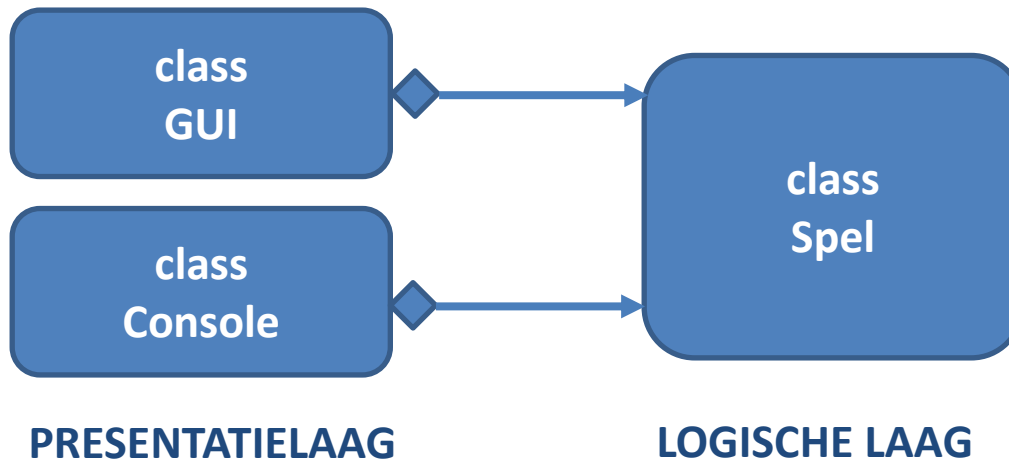


```
public class Auto {  
    //velden  
    private Stuur stuur; //compositie  
    private Wiel[] wielen; //compositie  
    private int kmStand; //geen compositie  
}
```

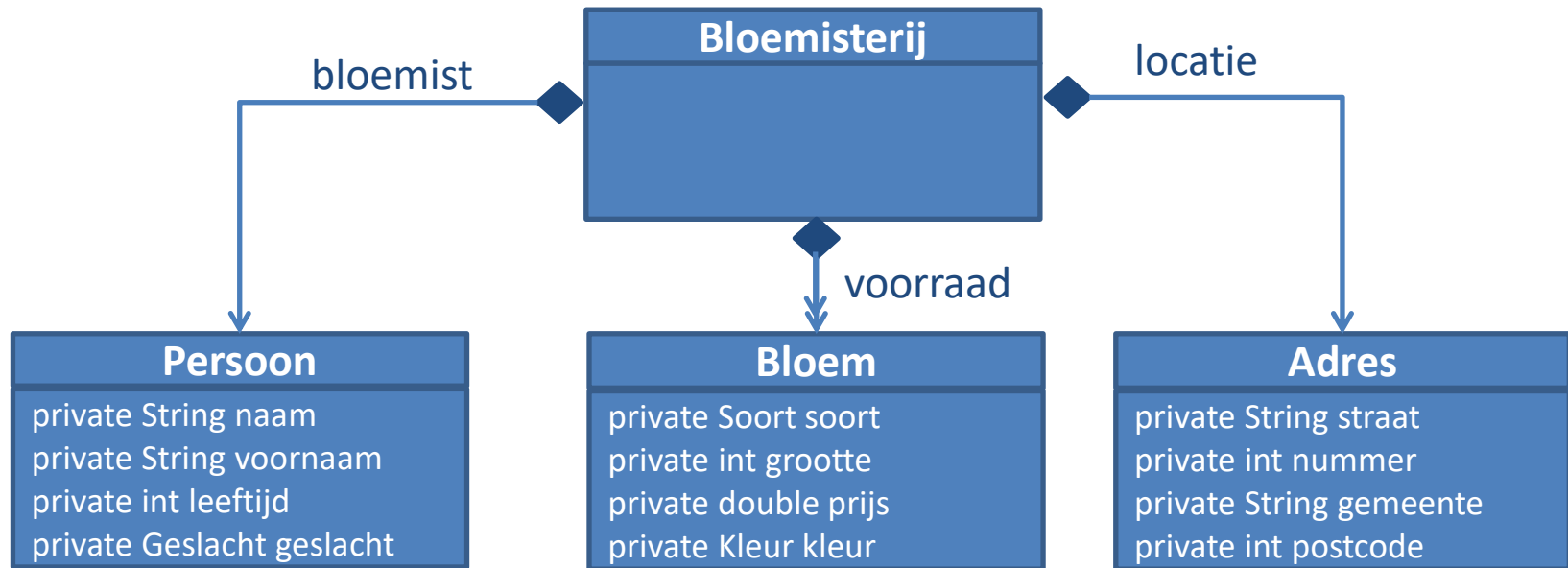
Ander concreet voorbeeld

```
public class GUI {  
    private Spel spel;  
}
```

```
public class Console {  
    private Spel spel;  
}
```



Uitgebreider voorbeeld



Een bloemisterij *wordt gekenmerkt door*:

- zijn **bloemist** = een persoon, i.e. object van de klasse Persoon
- zijn **voorraad** aan bloemen = array van Bloem-objecten
- zijn **locatie** = een adres, i.e. object van de klasse Adres

➔ Definieer deze klasse Bloemisterij, met zijn typische niet-standaard constructor

```
public class Bloemisterij {  
    private Persoon bloemist;  
    private Bloem[] voorraad;  
    private Adres locatie;  
  
    public Bloemisterij(Persoon p, Bloem[] bRij, Adres a) {  
        this.bloemist = p;  
        this.voorraad = bRij;  
        this.locatie = a;  
    }  
}
```

➔ Hoe maak je een concreet object van deze klasse Bloemisterij?

```
public static void main(String[] args) {  
    try {  
        Persoon persoon = new Persoon(..., ..., ..., ...);  
        Bloem[] bloemen = {new Bloem(..., ..., ..., ...), new Bloem(..., ..., ..., ...), ... };  
        Adres adres = new Adres(..., ..., ..., ...);  
  
        Bloemisterij bloemisterij = new Bloemisterij(persoon, bloemen, adres);  
  
    } catch(Exception e) {  
        System.out.println("Error: " + e.getMessage());  
    }  
}
```


Uitgebreider voorbeeld - Bis

Compositierelatie in zijn veld-voorstelling'

Garage
private Persoon garagist private Auto[] wagenpark private Adres locatie

Leest als:

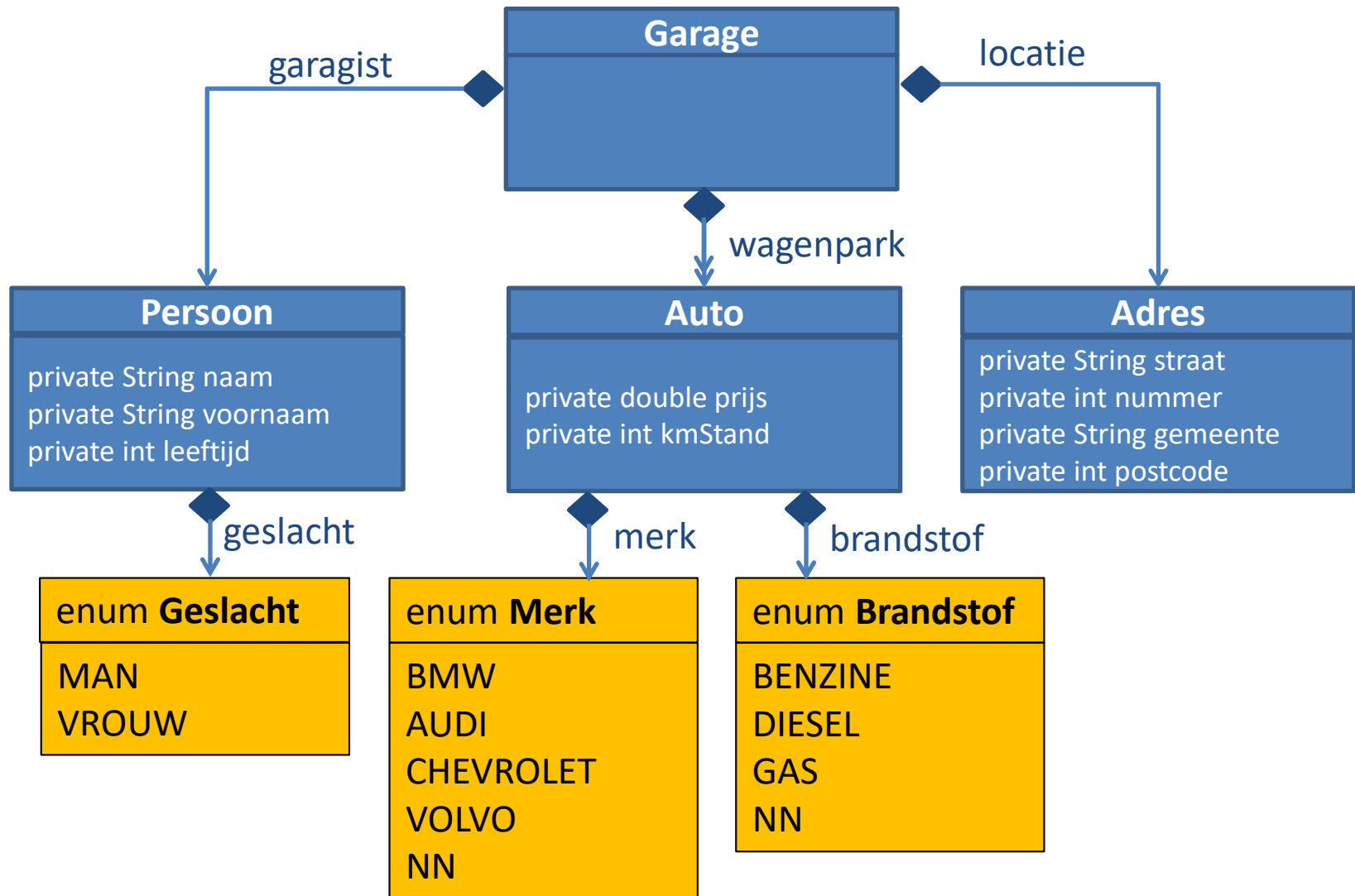
-Omvat ...

-Heeft een ...

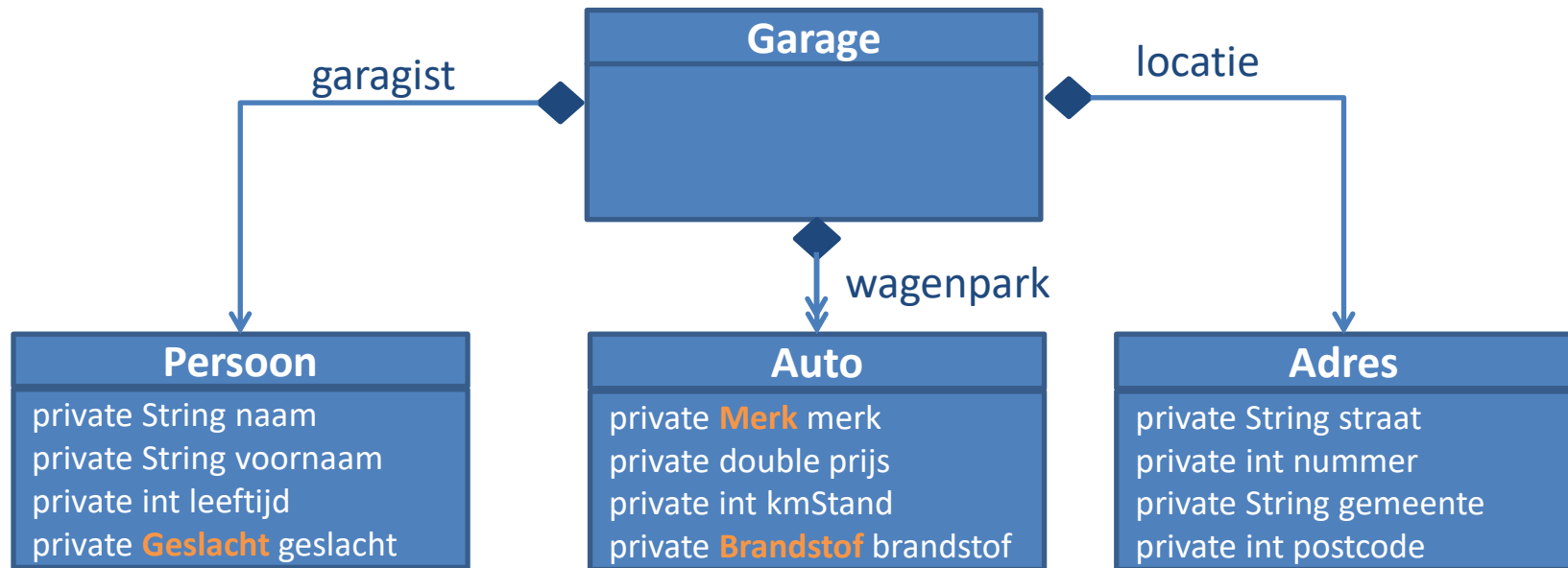
-Bestaat uit ...

-Wordt gekenmerkt door ...

Uitgebreider voorbeeld - Bis



UML – compositie- vs veldvoorstelling



Een garage wordt *gekenmerkt door*:

- zijn **garagist** = een persoon, i.e. object van de klasse Persoon
- zijn **wagenpark** = een array van Auto-objecten
- zijn **locatie** = een adres, i.e. object van de klasse Adres

➔ Definieer de klasse Garage, met zijn typische niet-standaard constructor

```
/**
 * Niet-standaard constructor voor een auto-object
 * @param merk een enumeratiewaarde uit enum Merk
 * @param prijs een double waarde die de prijs van een aangekochte auto voorstelt
 * @param km het aantal kilometer dat de auto bij aankoop op de teller heeft staan
 * @param brandstof een enumeratiewaarde uit enum Brandstof
 * @throws IllegalArgumentException exceptie die opgeworpen wordt wanneer ongeldige kenmerken aan een auto worden toegekend
 */
```

```
public class Auto {
    private Merk merk;
    private double prijs;
    private int kmStand;
    private Brandstof brandstof;

    public Auto() {
        this.merk = Merk.NN;
        this.brandstof = Brandstof.NN;
    }
}
```

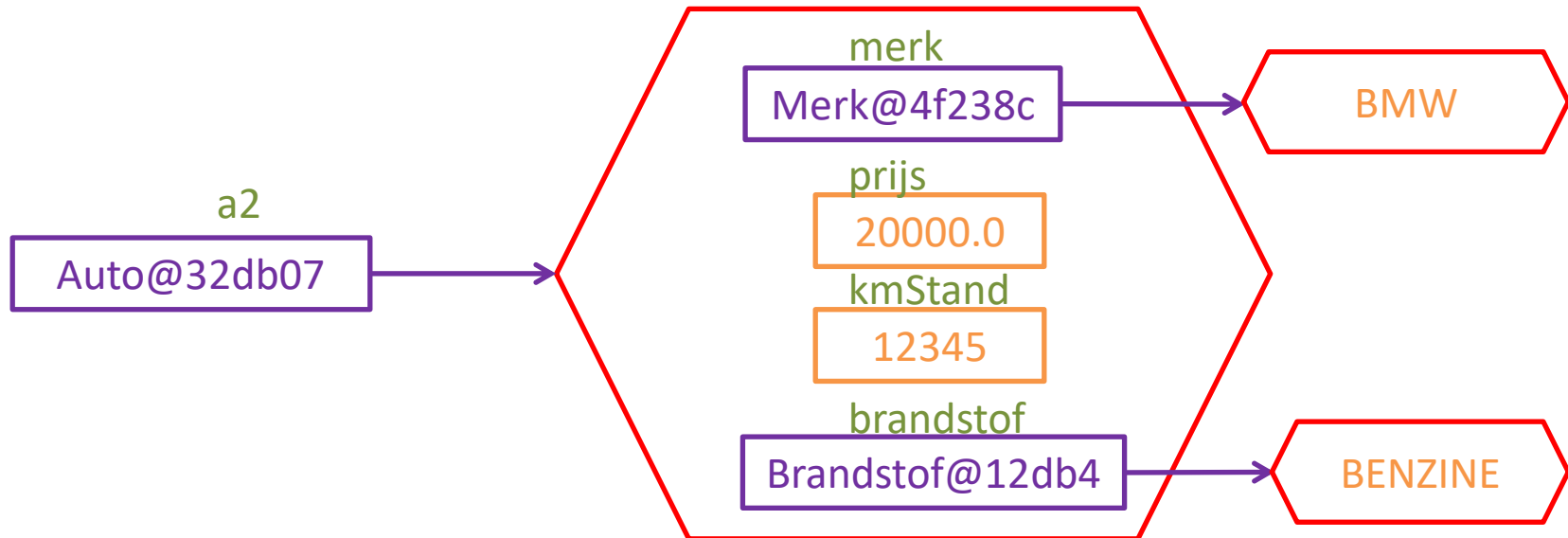
```
public enum Merk {
    BMW, AUDI, CHEVROLET, VOLVO, NN
}

public enum Brandstof {
    BENZINE, DIESEL, GAS, NN
}
```

```
public Auto(Merk merk, double prijs, int kmStand, Brandstof brandstof) {
    if (prijs < 0) throw new IllegalArgumentException("prijsen kunnen niet negatief zijn");
    if (km < 0) throw new IllegalArgumentException("kilometerstand kan niet negatief zijn");
    this.merk = merk;
    this.prijs = prijs;
    this.kmStand = kmStand;
    this.brandstof = brandstof;
}
}
```

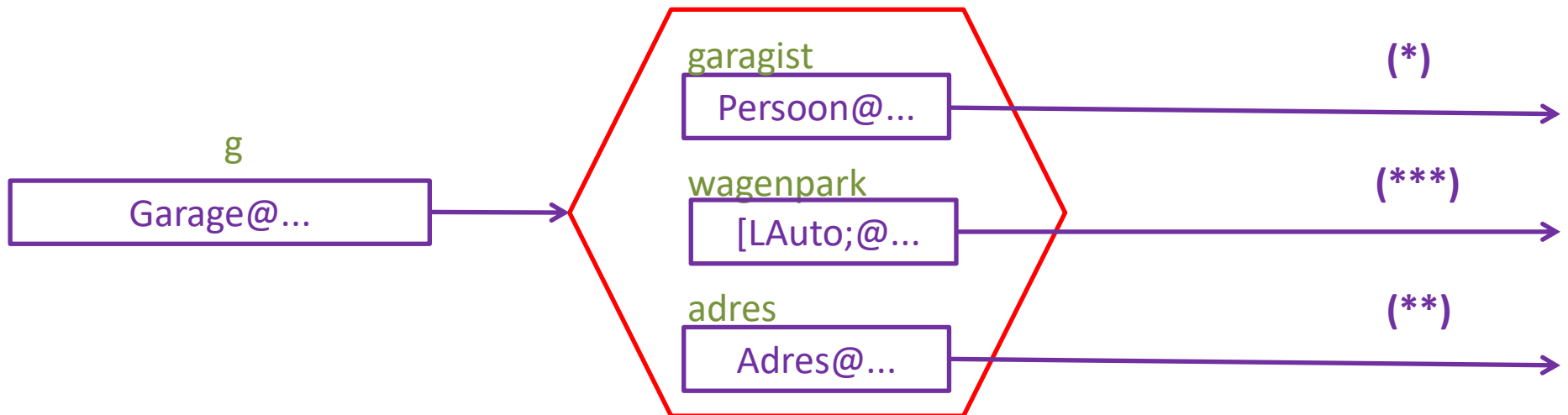
In presentatieklasse:

```
import logica.enumeraties.Brandstof;  
import logica.enumeraties.Merk;  
  
public class Console {  
    public static void main(String[] args) {  
        Auto a1 = new Auto();  
        Auto a2 = new Auto(Merk.BMW, 20000, 12345, Brandstof.BENZINE);  
    }  
}
```

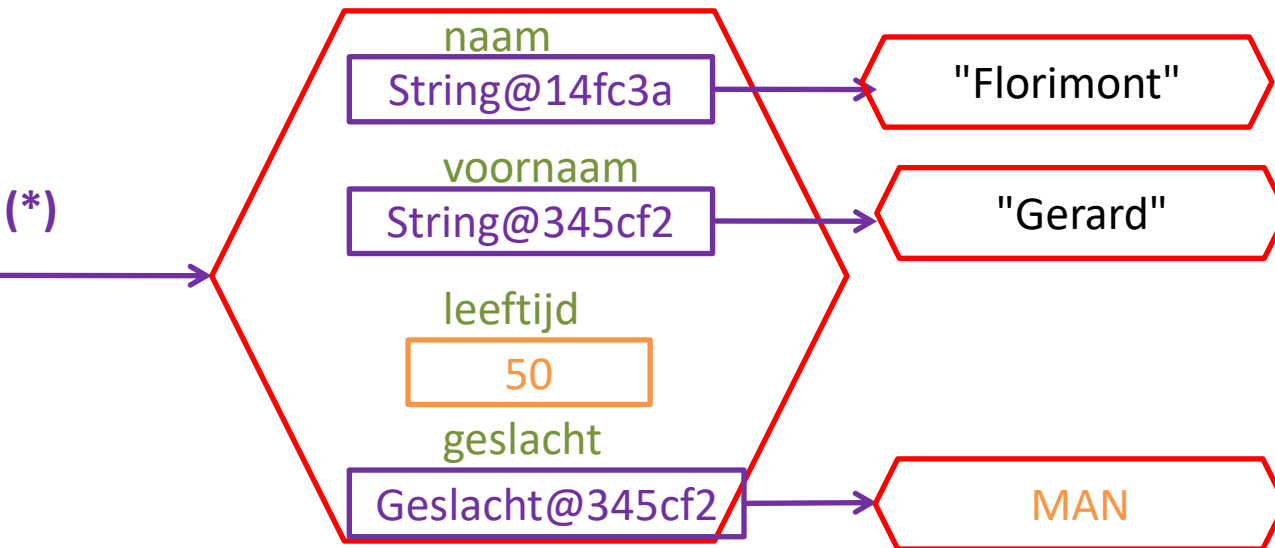


Voorbeeld v/e concrete garage (1/4)

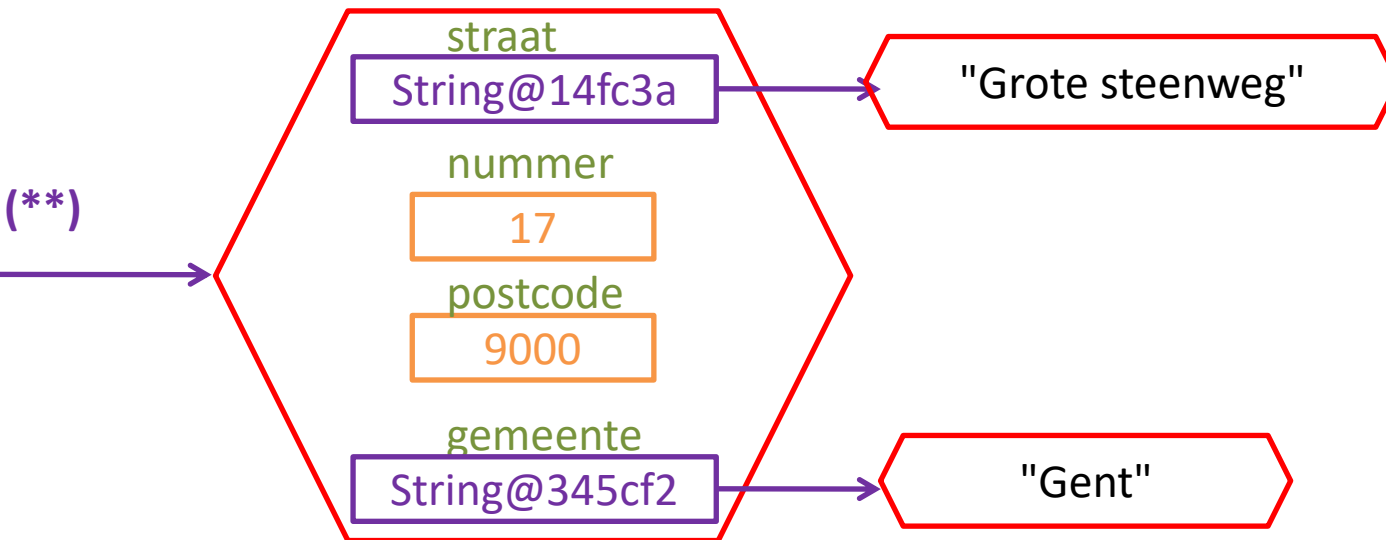
```
public class Garage {  
    private Persoon garagist;  
    private Auto[] wagenpark;  
    private Adres locatie;  
  
    public Garage(Persoon persoon, Auto[] autos, Adres adres) {  
        this.garagist = persoon;  
        this.wagenpark = autos;  
        this.locatie = adres;  
    }  
}
```



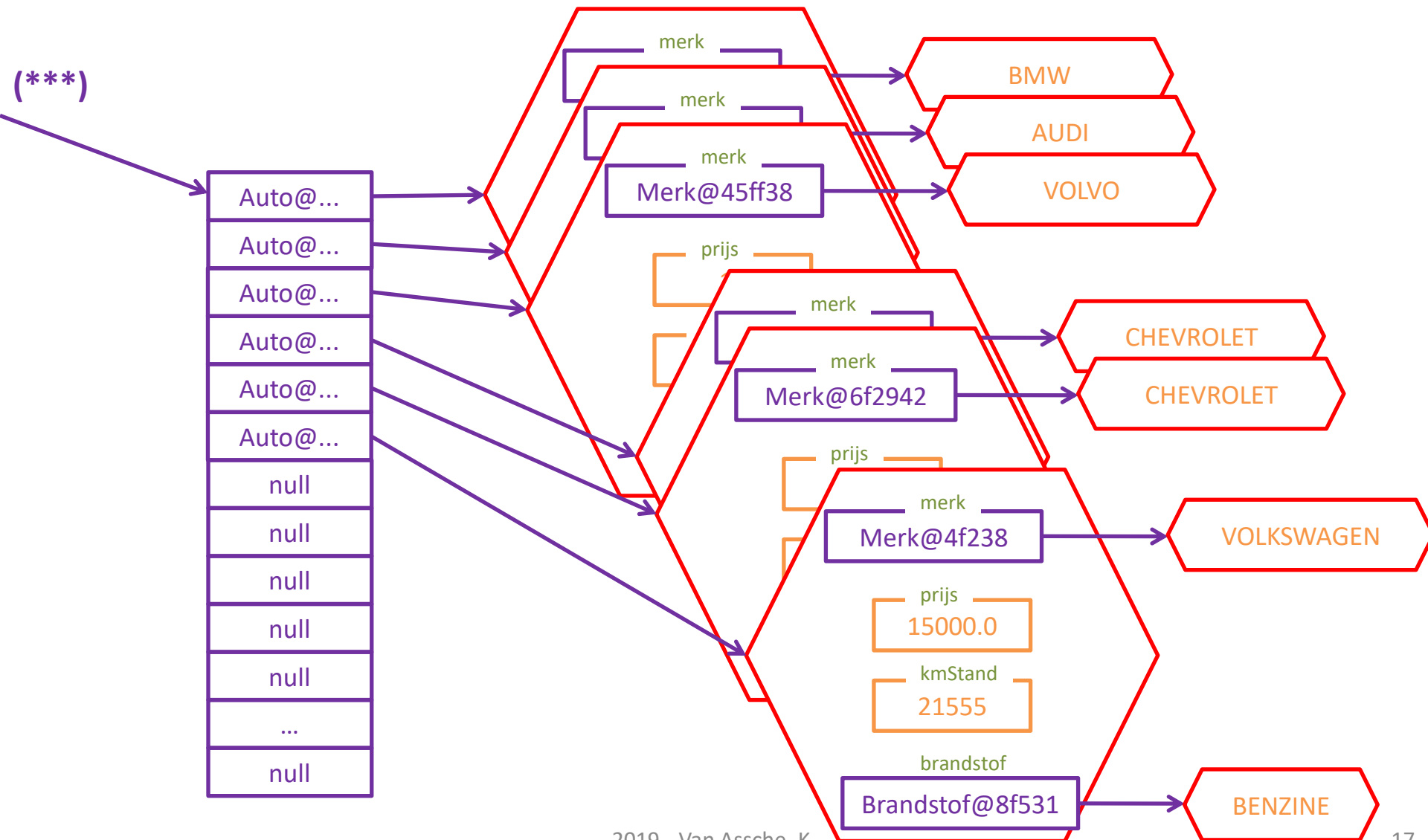
Voorbeeld v/e concrete garage (2/4)



Voorbeeld v/e concrete garage (3/4)



Voorbeeld v/e concrete garage (4/4)




Via welke code in je testprogramma realiseer je voorgaande 'concrete' realisatie?

```
public static void main(String[] args) {  
    try {  
        Persoon persoon = ...;  
        Auto[] autos = ...;  
        Adres adres = ...;  
        ...  
        Garage garage = new Garage(persoon, autos, adres);  
    } catch(Exception e) {  
        System.out.println("Error: " + e.getMessage());  
    }  
}
```

Concreet








```
public static void main(String[] args) {  
    try {  
        Persoon persoon = new Persoon("Florimont", "Gerard", 50, Geslacht.MAN);  
  
        Auto[] autos = new Auto[20];  
        autos[0] = new Auto(Merk.BMW, 30000.0, 0, Brandstof.BENZINE);  
        autos[1] = new Auto(Merk.AUDI, 35000.0, 0, Brandstof.DIESEL);  
        autos[2] = new Auto(Merk.VOLVO, 19500.0, 1100, Brandstof.BENZINE);  
        autos[3] = new Auto(Merk.CHEVROLET, 18000.0, 52148, Brandstof.BENZINE);  
        autos[4] = new Auto(Merk.CHEVROLET, 19999.0, 22121, Brandstof.DIESEL);  
        autos[5] = new Auto(Merk.VOLKSWAGEN, 15000, 21555, Brandstof.BENZINE);  
  
        Adres adres = new Adres("Grote steenweg", 17, 9000, "Gent");  
  
        Garage garage = new Garage(persoon, autos, adres);  
  
    } catch(Exception e) {  
        System.out.println("Error: " + e.getMessage());  
    }  
}
```

Fix all imports

 Fix All Imports ✕

Select the fully qualified name to use in the import statement.

Import Statements:

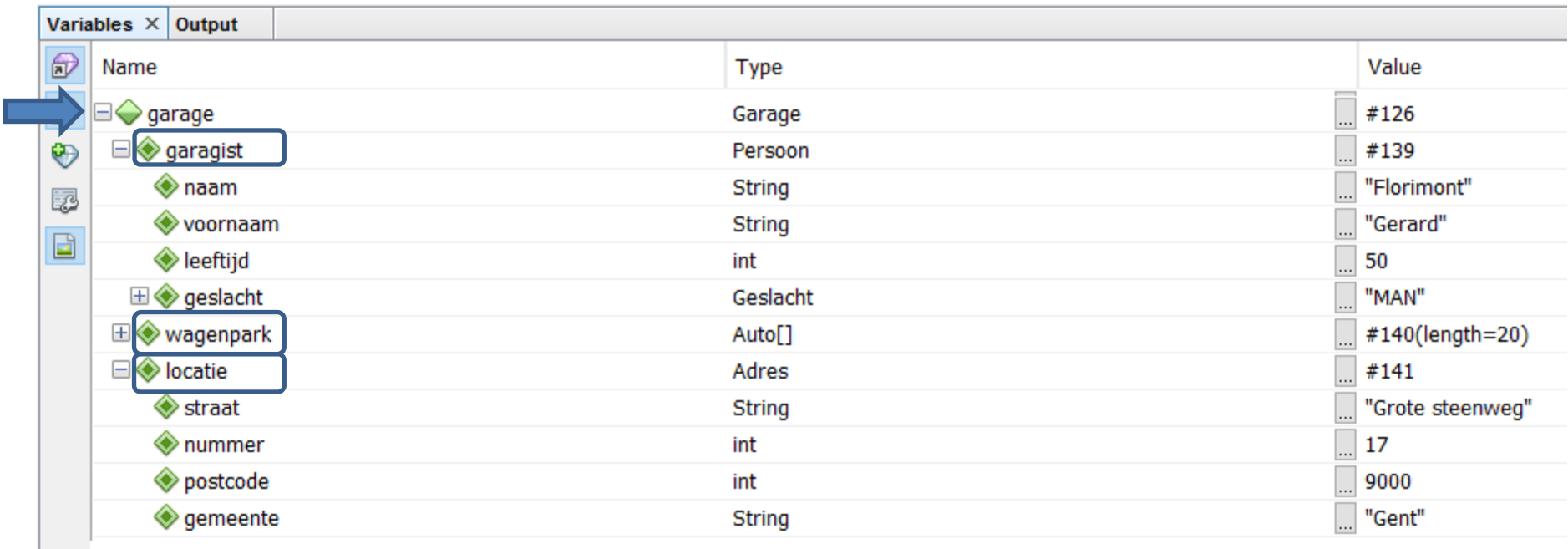
Persoon	 logica.Persoon	▼
Geslacht	 logica.enumeraties.Geslacht	▼
Auto	 logica.Auto	▼
Garage	 logica.Garage	▼
Merk	 logica.enumeraties.Merk	▼
Brandstof	 logica.enumeraties.Brandstof	▼
Adres	 logica.Adres	▼

☒ Remove unused imports

OK

Cancel

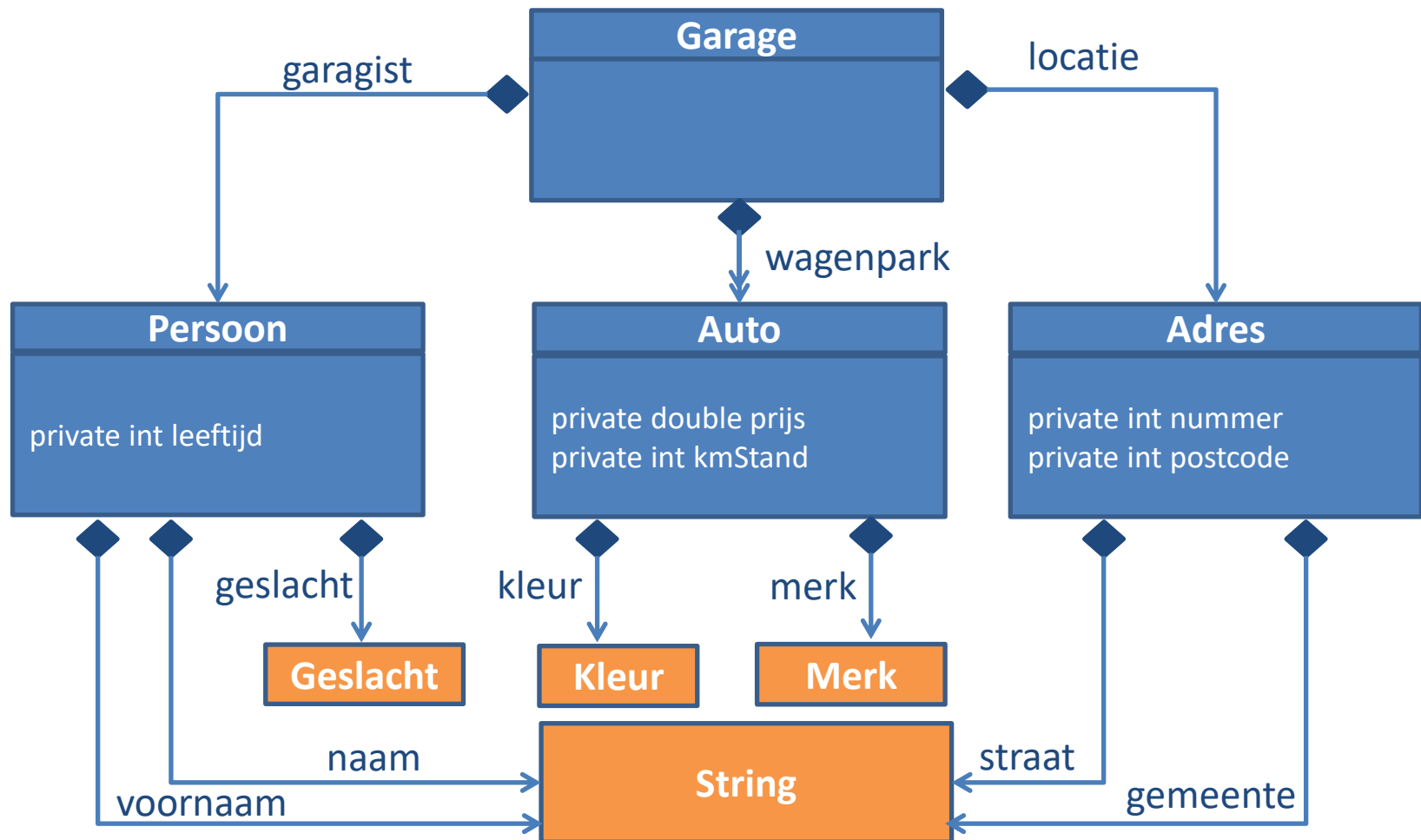
Variables venster in Netbeans



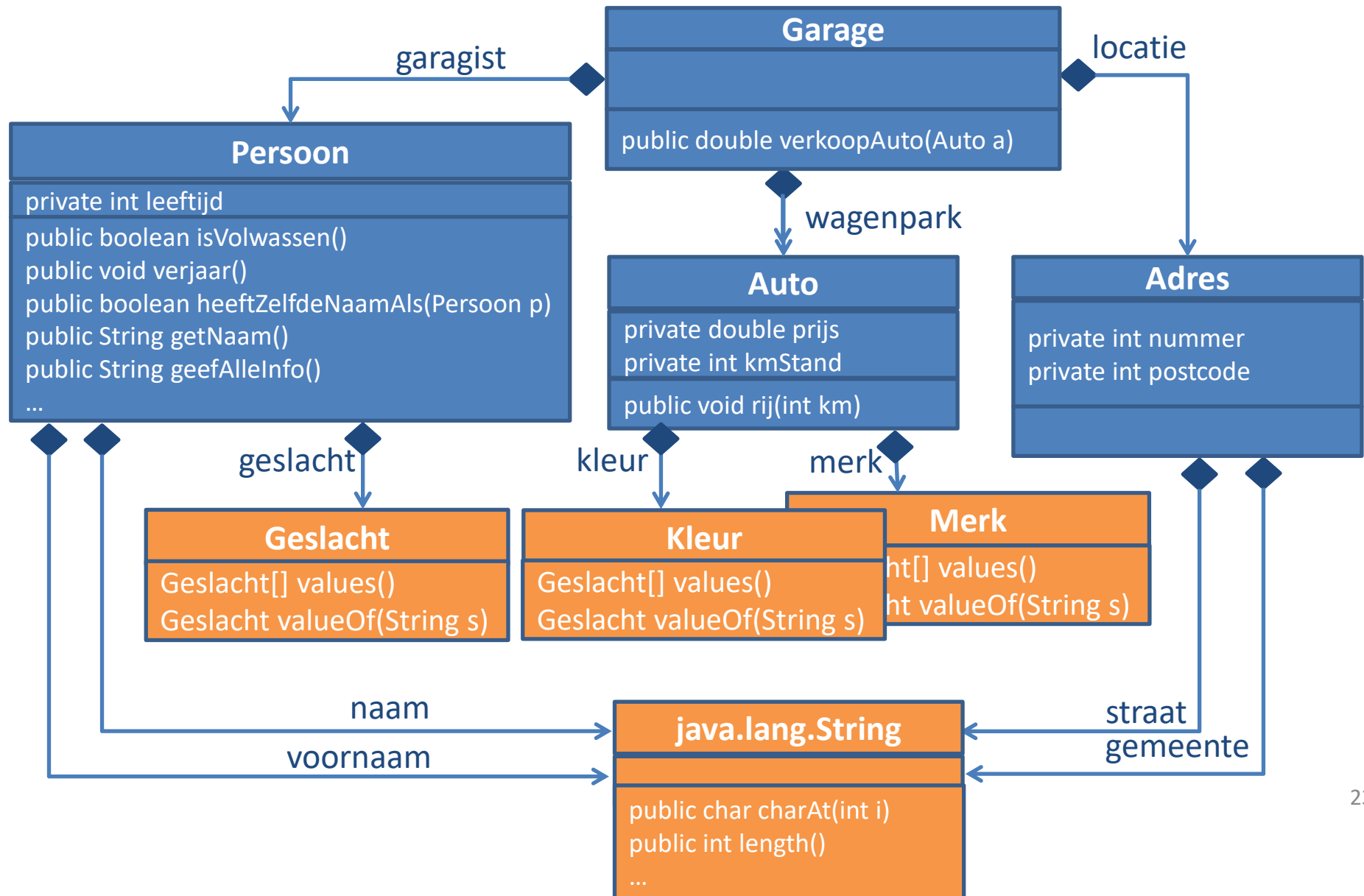
Variables X		Output	
Name	Type	Value	
garage	Garage	#126	
garagist	Persoon	#139	
naam	String	"Florimont"	
voornaam	String	"Gerard"	
leeftijd	int	50	
geslacht	Geslacht	"MAN"	
wagenpark	Auto[]	#140(length=20)	
locatie	Adres	#141	
straat	String	"Grote steenweg"	
nummer	int	17	
postcode	int	9000	
gemeente	String	"Gent"	

Klasse Garage

‘uitgebreide associatie-voorstelling’



Met aanduiding van methoden



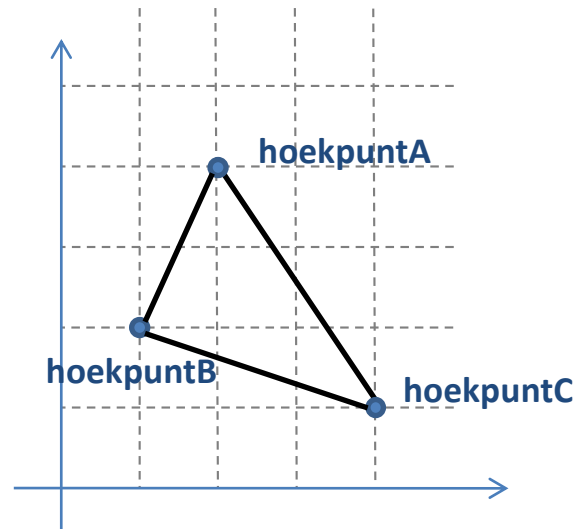
Enkele Oefeningen op compositie

Hergebruik van code via Compositie

OEFENING

Teken het UML-klassendiagramma waarin de relatie tussen volgende klassen wordt aangegeven:

- Punt
- Lijn
- Driehoek
- Cirkel
- Rechthoek
- Vierkant



TIP: Denk in termen van compositie:

Bv. Een lijn heeft een begin- en eindpunt, een driehoek heeft 3 hoekpunten, een cirkel wordt gekenmerkt door een middelpunt en een straal, bij een vierkant heeft elke zijde dezelfde lengte, ...

Bepaal data en de functionaliteit

- Wat is de gevraagde functionaliteit ?
 - Bv. omtrek en/of oppervlakte bepalen
 - Moet je de figuur ook op een specifieke plaats kunnen lokaliseren t.o.v. een X- en Y-as (2-dim vlak) ?
 - Moet je de figuur kunnen tekenen in een vlak?

=> Gewenste functionaliteit bepaalt de data die in het object bewaard moet worden

Vierkant

A. Omtrek en oppervlakte kunnen bepalen

```
public int berekenOmtrek() {  
    return zijde * 4;  
}
```

```
public int berekenOppervlakte() {  
    return Math.pow(zijde, 2);  
}
```

Vierkant

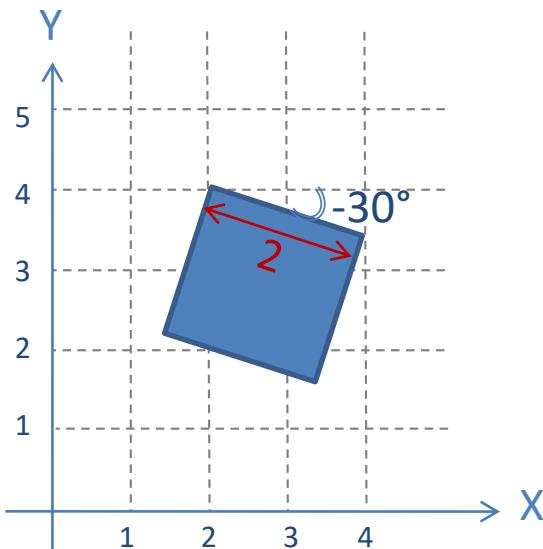
- int zijde

+ Vierkant(int zijde)
+ int berekenOmtrek()
+ int berekenOppervlakte()

```
Vierkant v = new Vierkant(2);
```

B. Kunnen localiseren t.o.v. een X- en Y-as

C. Kunnen tekenen in een vlak



Punt

- int x
- int y

linksBovenPunt

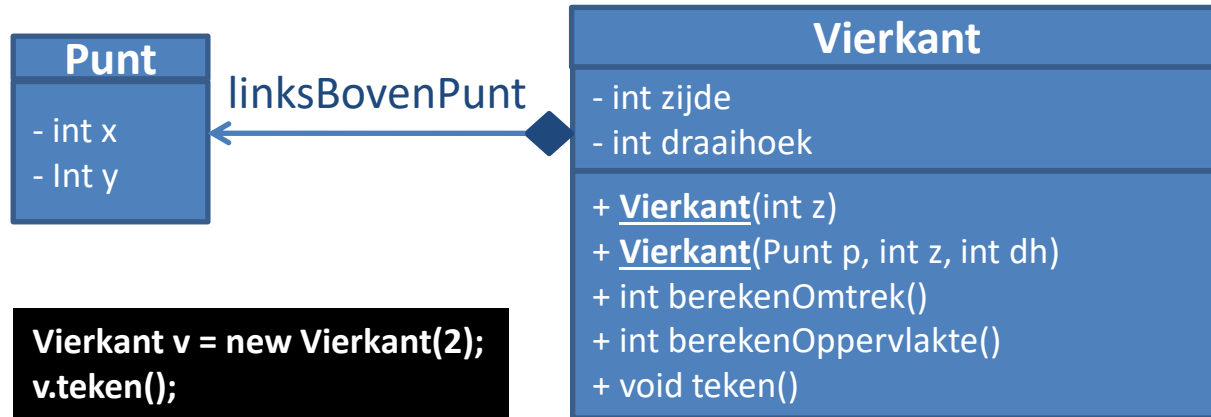
Vierkant

- int zijde
- int draaihoek

+ Vierkant(Punt p, int z, int dh)
+ int berekenOmtrek()
+ int berekenOppervlakte()
+ void teken()

```
Vierkant v = new Vierkant(new Punt(2,4), 2, -30);  
v.teken();
```

Logische klasse



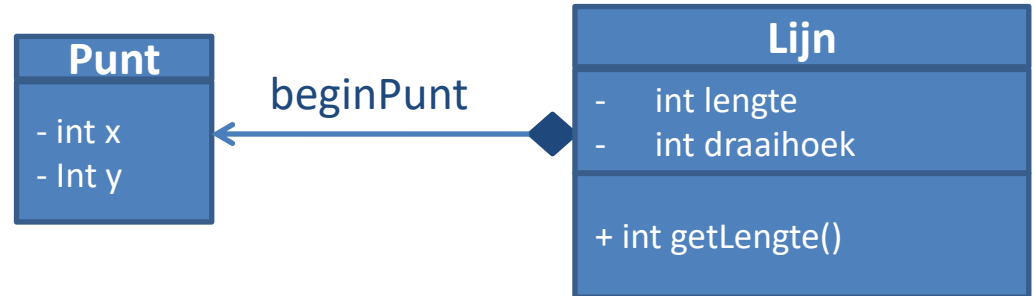
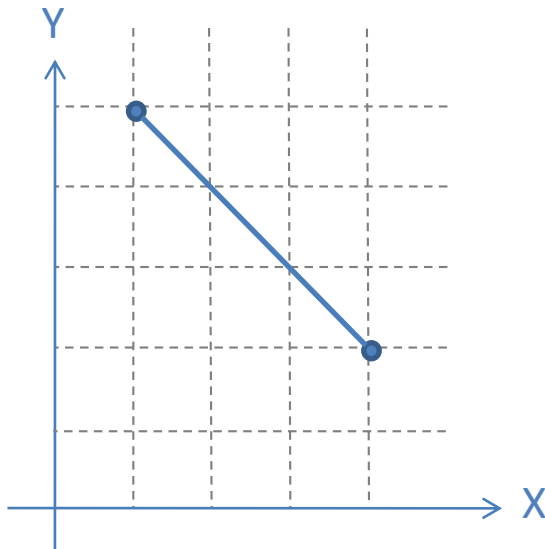
Wat zal er concreet gebeuren?

- Een vierkant met linkerbovenhoek in de oorsprong van het assenstelsel (default) ?
- Null pointer exceptie - wegens ontbrekend linksbovenpunt ?
- Niets, net omwille van de ontbrekende data?

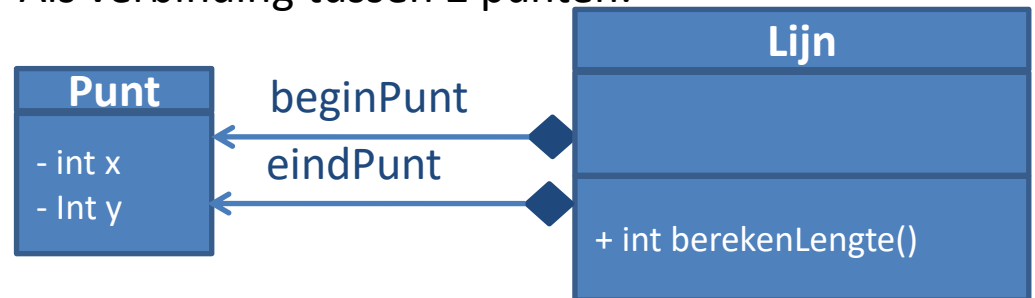
```
public void teken() {  
    if (linksBovenPunt != null) {  
        //teken  
    }  
}
```

Lijn

Start op bepaalde plaats en heeft lengte en richting:

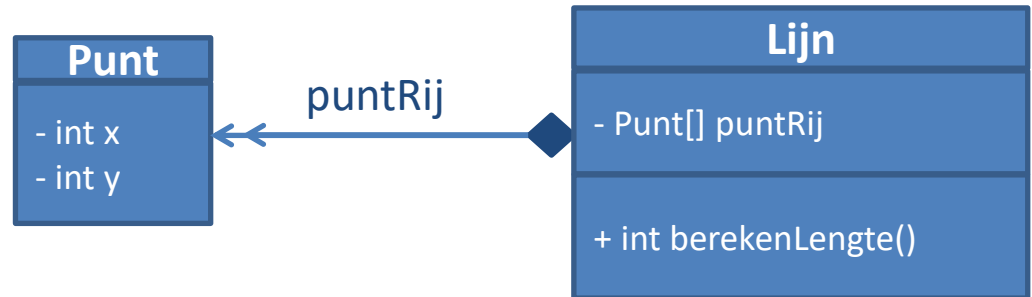
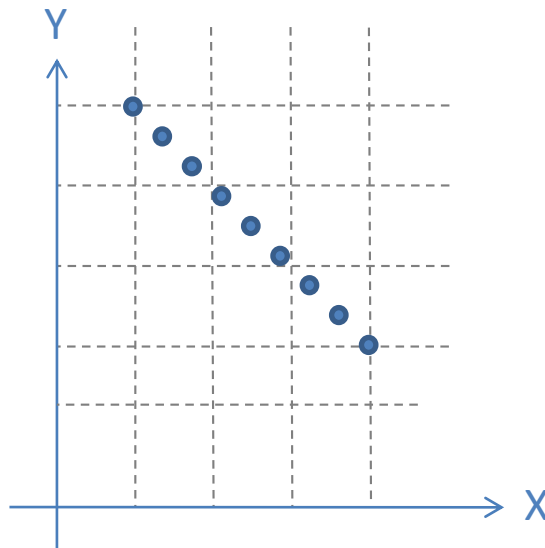


Als verbinding tussen 2 punten:



```
public int berekenLengte() {
    return Math.sqrt( Math.pow(eindPunt.x - beginPunt.x, 2) +
        Math.pow(eindPunt.y - beginPunt.y, 2) );
}
```

Lijn als collectie van discrete punten

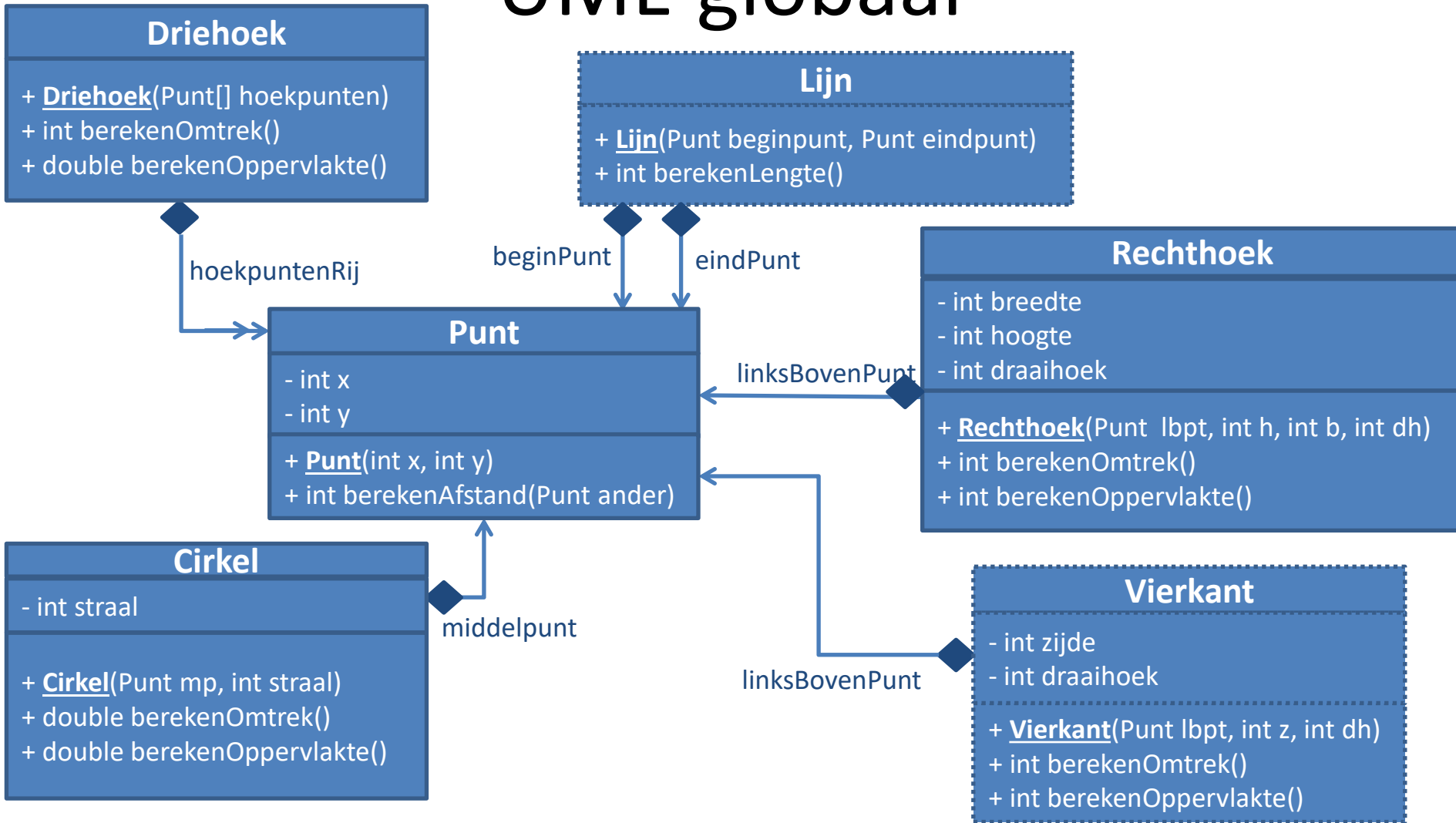


```
public int berekenLengte() {
    return Math.sqrt( Math.pow(puntRij[puntRij.length-1].x - puntRij[0].x, 2) +
        Math.pow(puntRij[puntRij.length-1].y - puntRij[0].y, 2) );
}
```

```
public int berekenLengte() {
    return puntRij[puntRij.length-1].berekenAfstand(puntRij[0]);
}
```

Cf. methode in klasse Punt:
public int berekenAfstand(Punt anderPunt)

UML-globaal



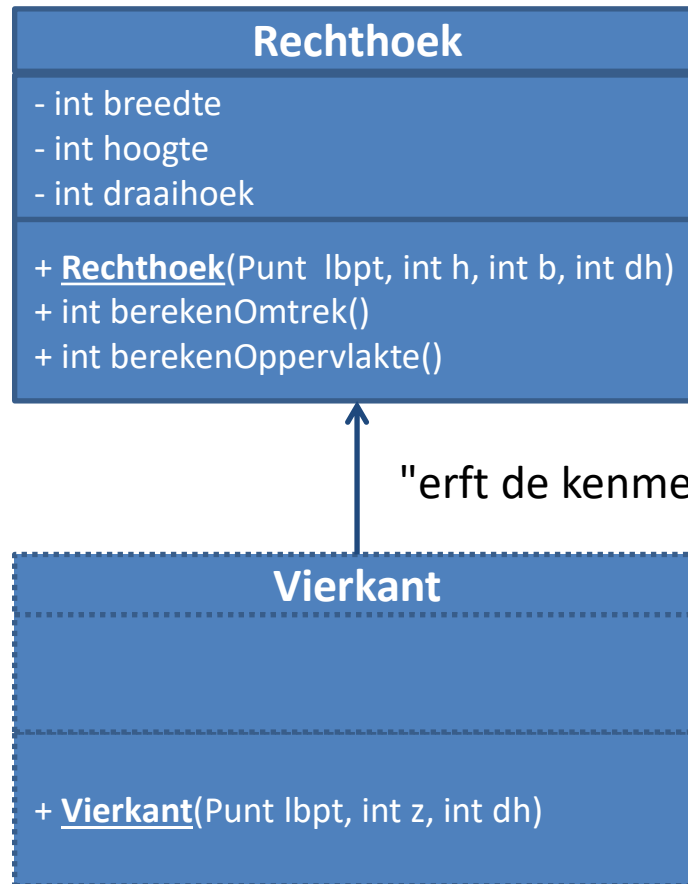
EXTRA OPGAVE-VOORBEELDEN

- Gezin
 - Gezin heeft gezinsleden
 - Gezin woont op bepaald adres
 - Van elk gezinslid hou je naam, leeftijd en geslacht bij
- Parking
 - Parking heeft naam en aantal parkeerplaatsen voor wagens
 - Parkeerplaatsen kunnen al dan niet voorbehouden zijn
 - Elke wagen heeft een eigenaar en een nummerplaat
- Postkantoor
 - Postkantoor heeft naam en adres
 - Postbodes verdelen de post van het postkantoor naar de gewenste bestemming
- Cursusdienst
 - Beheert de aankoop van cursussen voor studenten
 - Studenten kopen cursussen volgens hun opleiding en studieprogramma

Hergebruik van code



Vierkant is een 'bijzonder' geval van een rechthoek

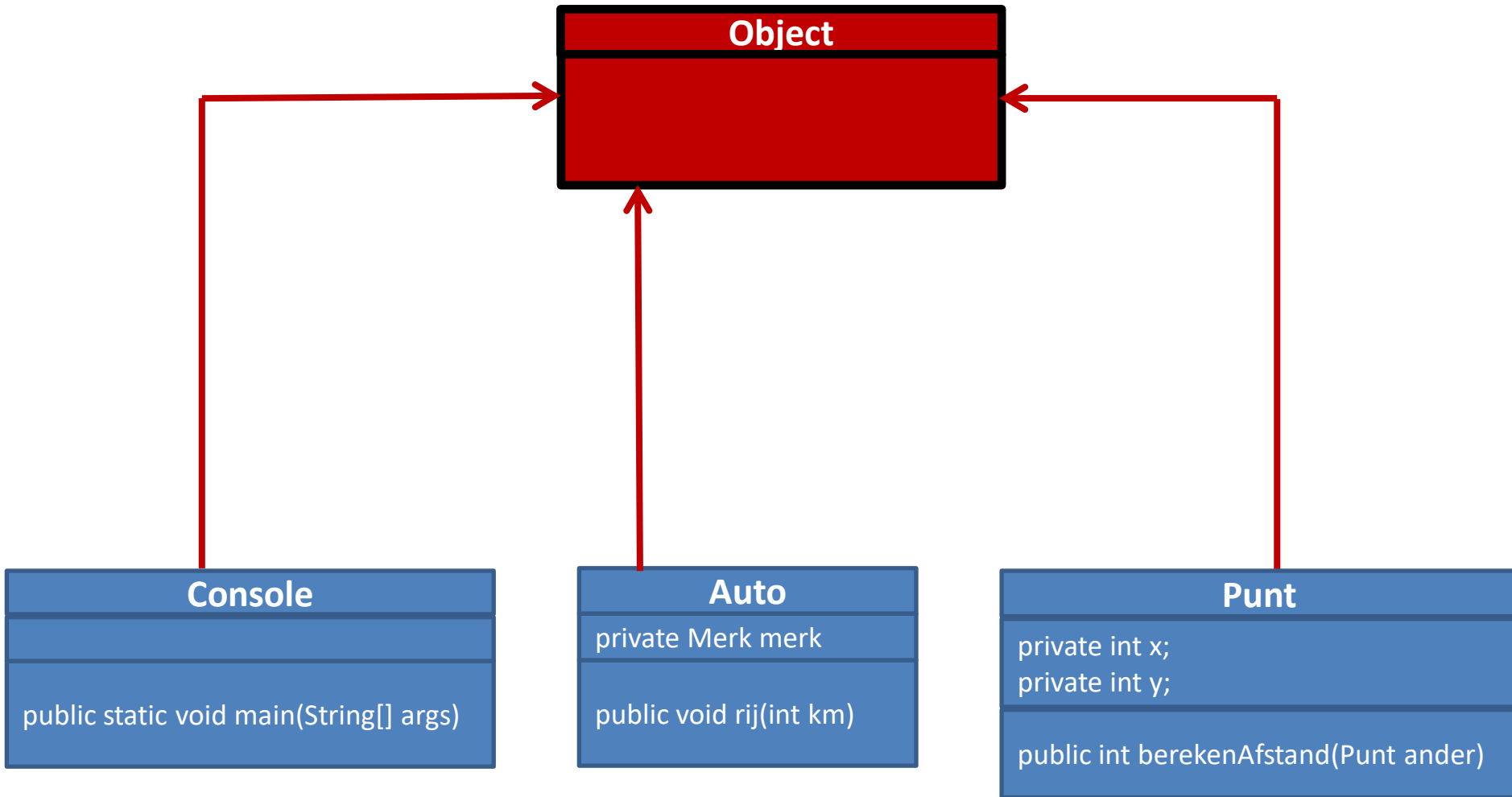


Zie volgende les

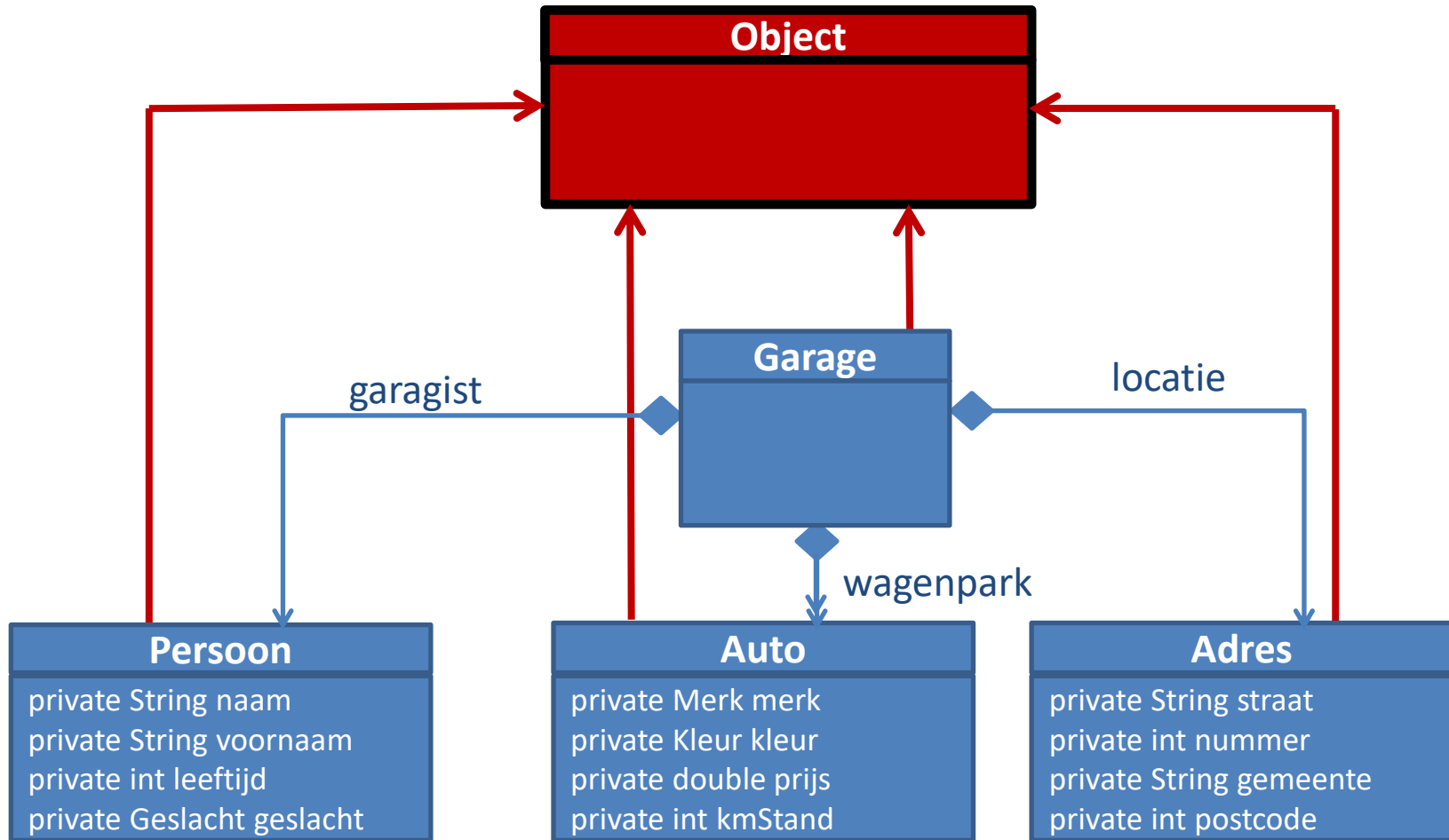
Hergebruik van code



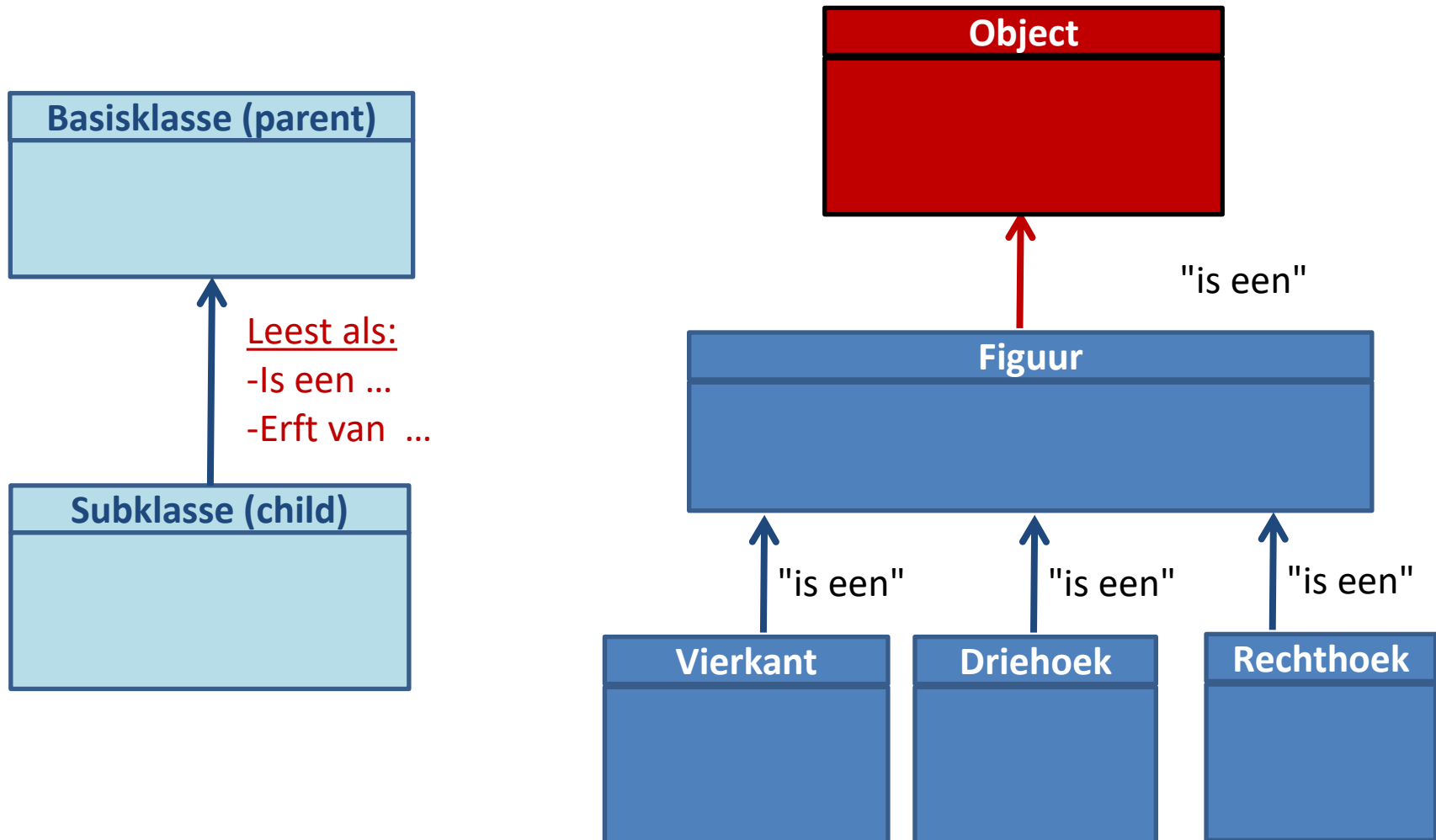
ALLE klassen erven over van dé root klasse Object



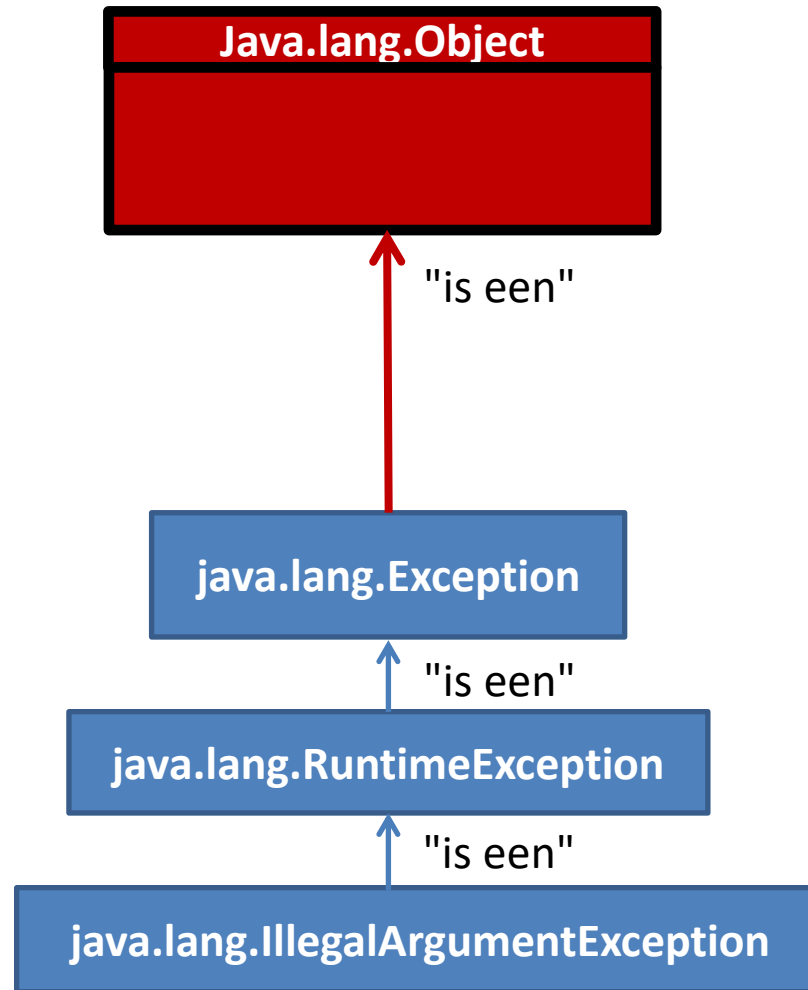
ALLE klassen erven over van dé root klasse Object



Terminologie & voorbeeld



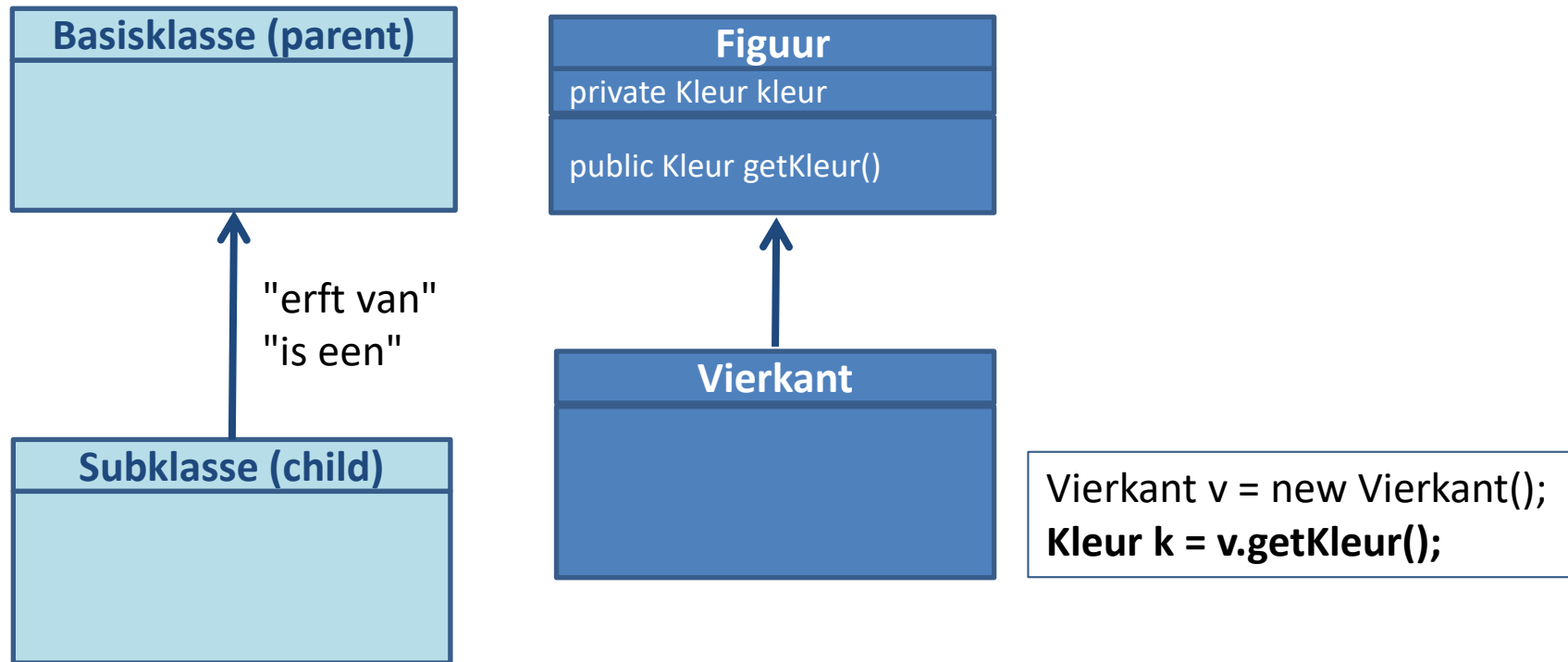
Zo ook voor excepties



Wát erf je dan?

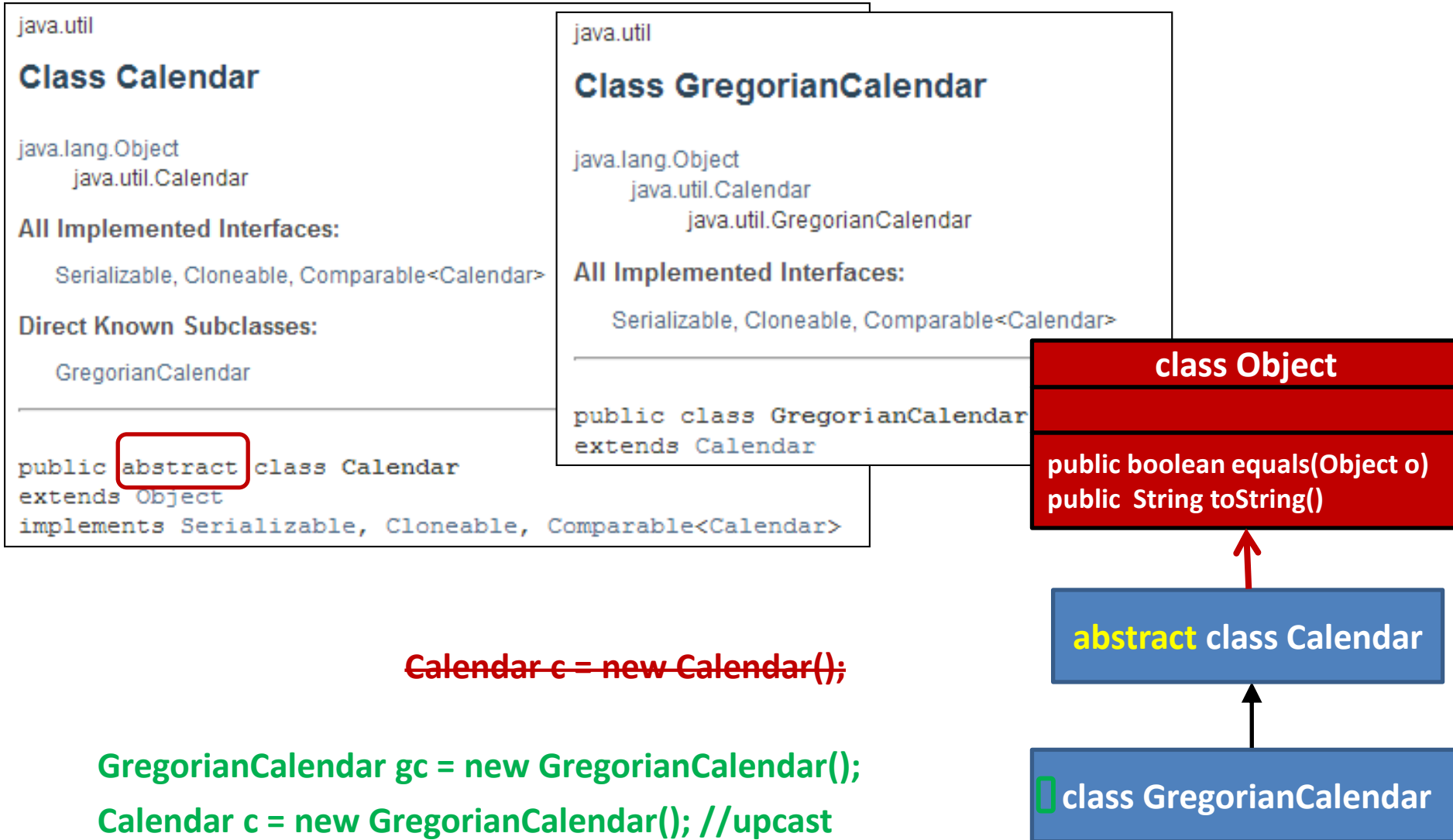
Kenmerken en functionaliteit v/d basisklasse

- Kenmerken → velden
- Functionaliteit → methoden



abstracte klasse

Geen objecten/instanties mogelijk van deze klasse!



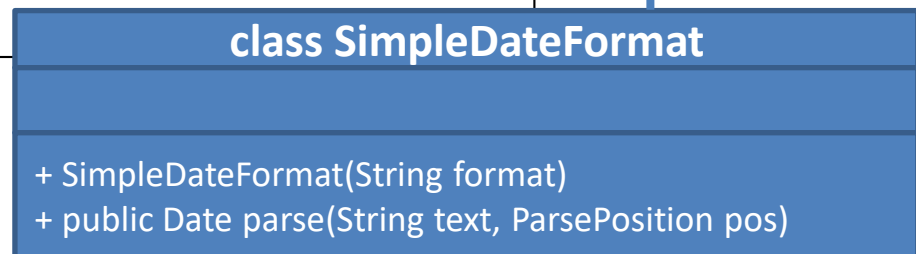
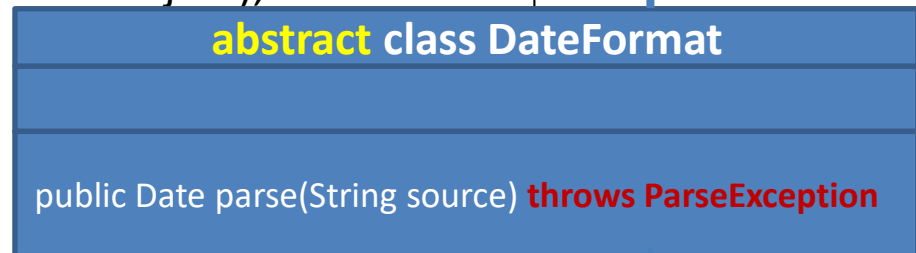
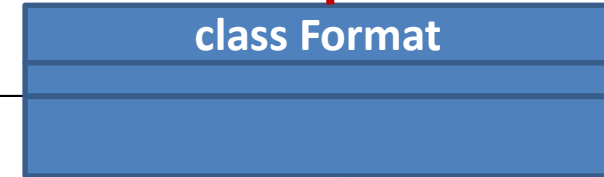
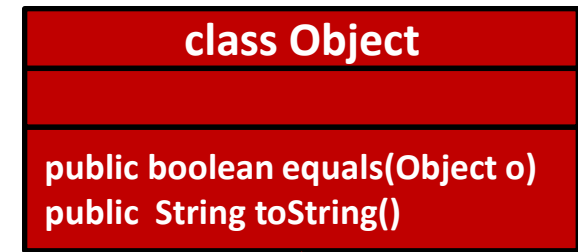
Vb: SimpleDateFormat

```
java.lang.Object
  java.text.Format
    java.text.DateFormat
      java.text.SimpleDateFormat
```

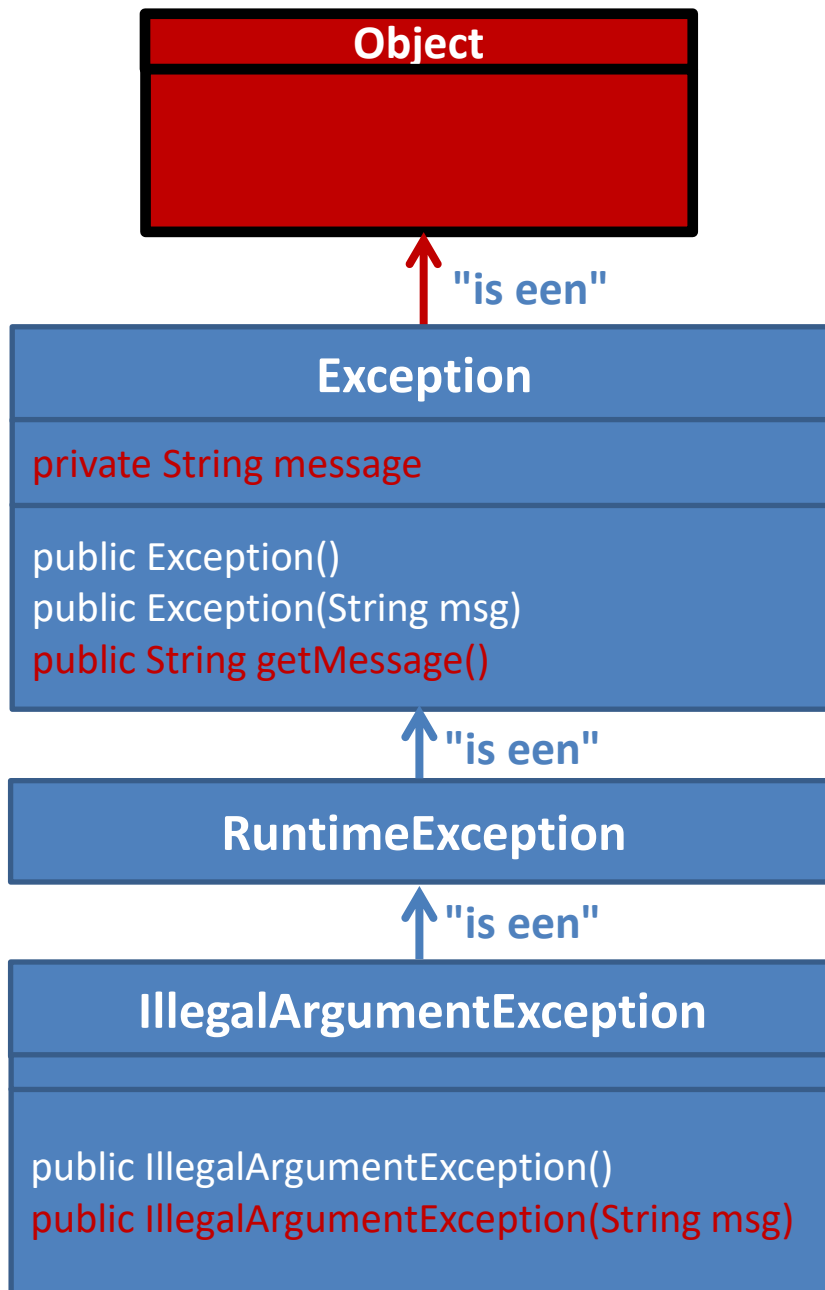
```
java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.text.ParseException
```

**checked exception*

```
private boolean isGeldig() {
    try {
        SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
        sdf.parse(this.dag + "-" + this.maand + "-" + this.jaar);
        return true;
    } catch (ParseException e) {
        System.out.println(e.getMessage());
        return false;
    }
}
```



Het is de 'parse' methode van de parent klasse DateFormat die hier wordt opgeroepen!!



java.lang

Class Object

java.lang.Object

Dé root van ALLE klassen

```
public class Object
```

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

See Also:

[Class](#)

Constructor Summary

[Object](#)()

Method Summary

protected Object	clone () Creates and returns a copy of this object.
boolean	equals (Object obj) Indicates whether some other object is "equal to" this one.
protected void	finalize () Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class <?>	getClass () Returns the runtime class of this <code>Object</code> .
int	hashCode () Returns a hash code value for the object.
void	notify () Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll () Wakes up all threads that are waiting on this object's monitor.
String	toString () Returns a string representation of the object.

Object

**public boolean equals(Object o)
public String toString()**

Signatuur van de 'root' methoden

- toString `public String toString()`
- equals `public boolean equals (Object o)`

basisvoorbeeld

Object

```
public boolean equals(Object o)
public String toString()
```

Hond

```
public class Hond {
}
```

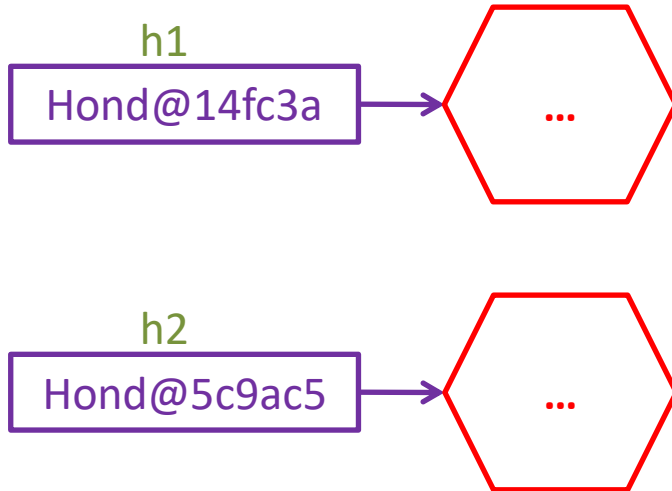
```
public class Console {
    Hond h1 = new Hond();
    System.out.println( h1.toString() );    = System.out.println(h1);

    Hond h2 = new Hond();
    System.out.println( h2.toString() );    = System.out.println(h2);

    if ( h1.equals(h2) ) {
        System.out.println("gelijke object referentie");
    }
    else {
        System.out.println("verschillende object referentie");
    }
}
```

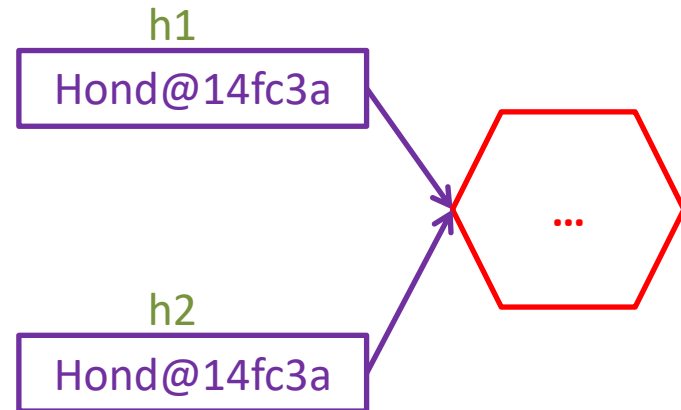
Concrete output

```
Hond h1 = new Hond();  
Hond h2 = new Hond();
```



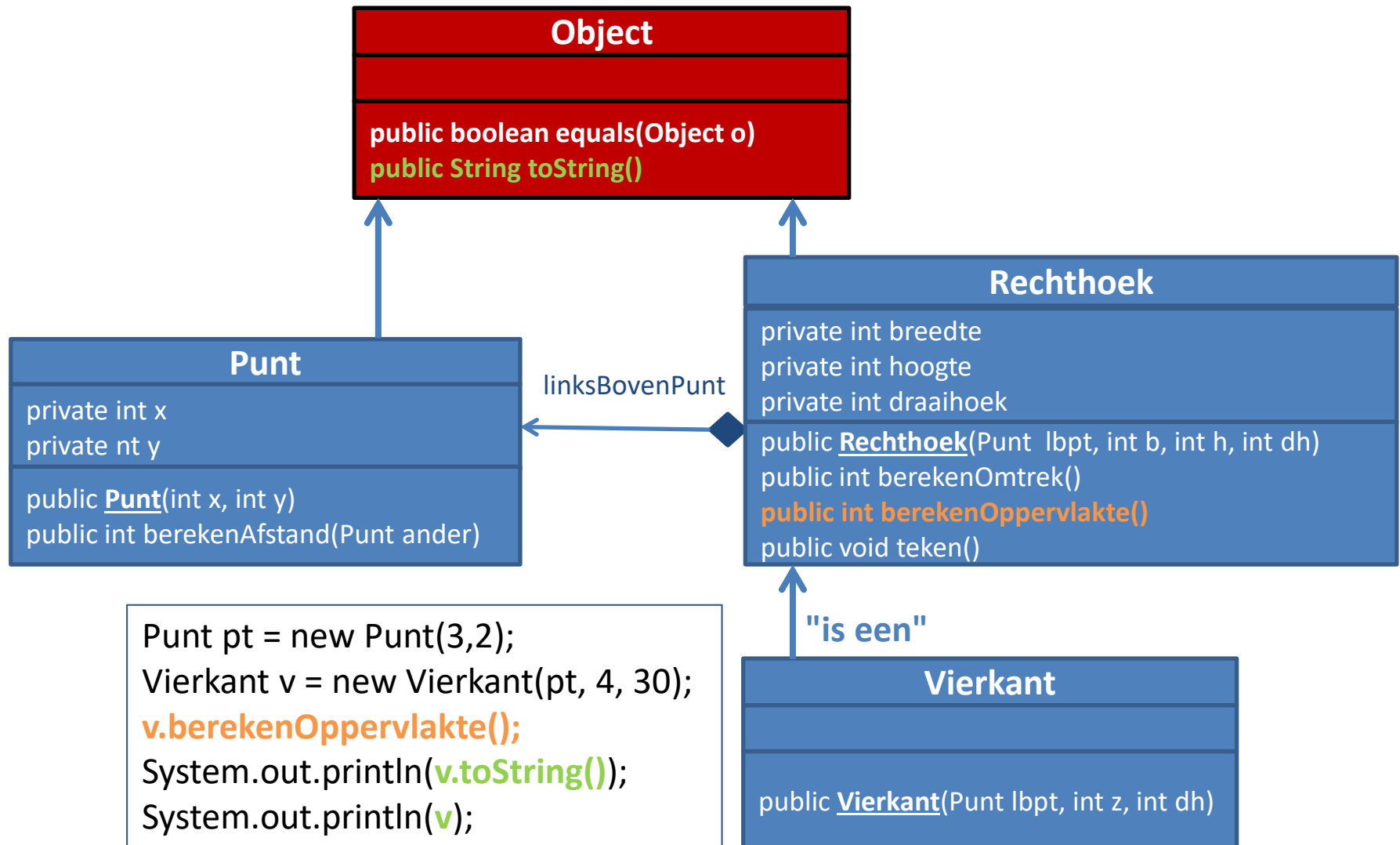
Hond@14fc3a
Hond@5c9ac5
Verschillende object referentie

```
Hond h1 = new Hond();  
Hond h2 = h1;
```

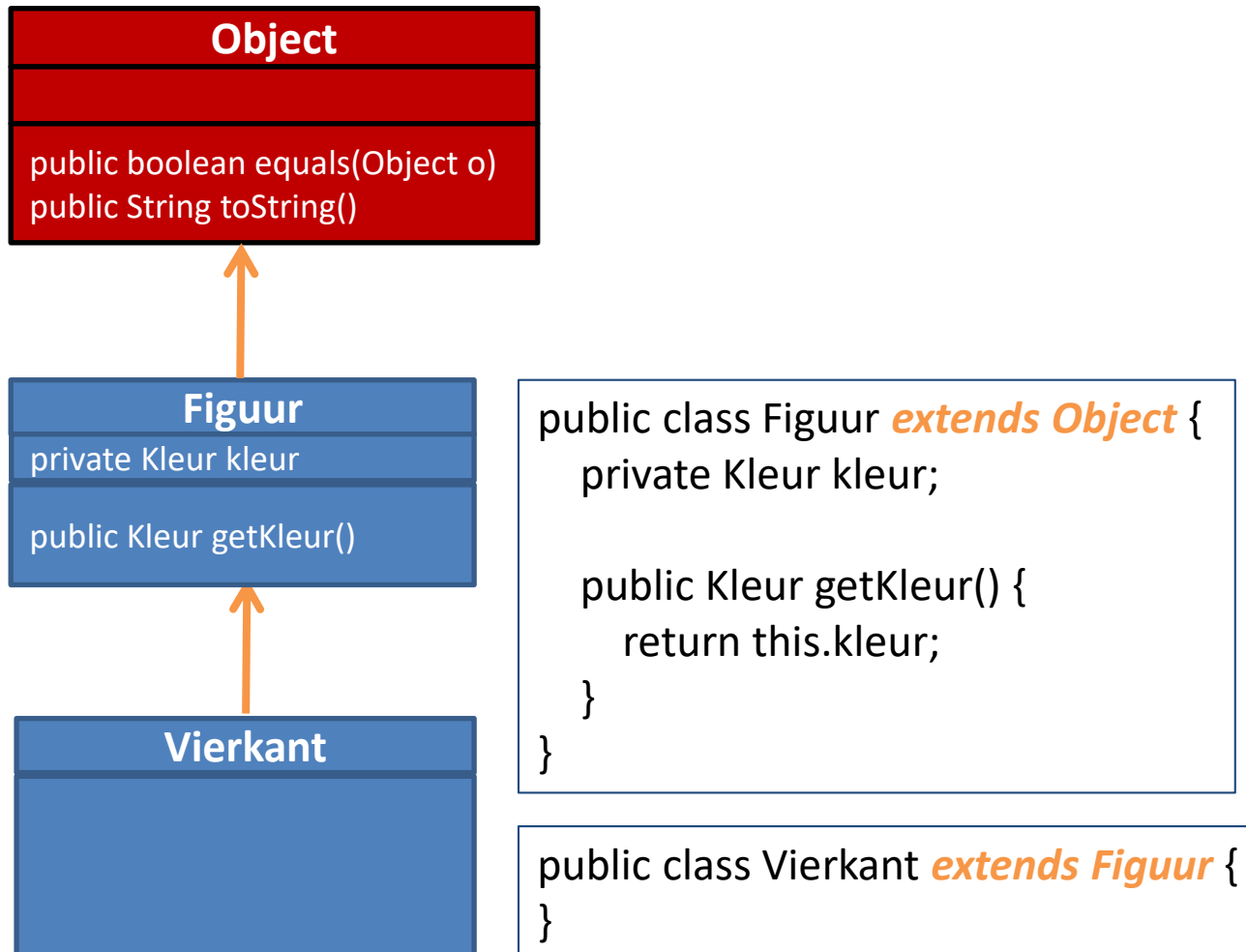


Hond@14fc3a
Hond@14fc3a
Gelijke object referentie

Vierkant als bijzonder geval v/e rechthoek



UML klassendiagramma <> code



Élke klasse *erft van* de 'root' klasse Object


```
public class Figuur extends Object {  
}
```

```
public class Figuur extends java.lang.Object {  
}
```


```
public class Figuur {  
}
```

(default)

Élke Java klasse erft van de 'root' klasse Object *-direct- of -indirect-*



```
java.lang.Object  
└─ java.lang.String
```

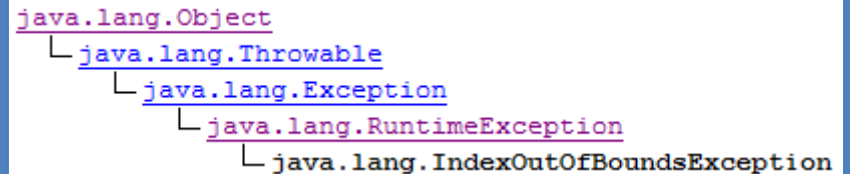


```
java.lang.Object  
└─ java.lang.Number  
   └─ java.lang.Integer
```

```
java.lang.Object  
└─ java.lang.Math
```

```
java.lang.Object  
└─ java.util.Calendar  
   └─ java.util.GregorianCalendar
```

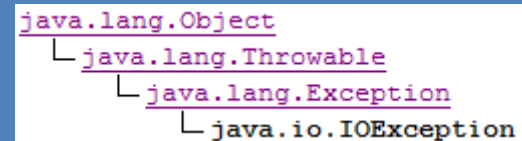
```
java.lang.Object  
└─ java.util.Formatter
```



```
java.lang.Object  
└─ java.lang.Throwable  
   └─ java.lang.Exception  
      └─ java.lang.RuntimeException  
         └─ java.lang.IndexOutOfBoundsException
```

i.e. unchecked exception

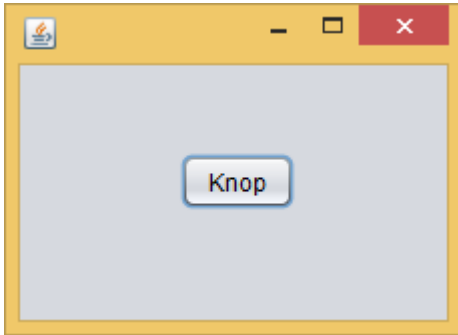
```
java.lang.Object  
└─ java.util.Scanner
```



```
java.lang.Object  
└─ java.lang.Throwable  
   └─ java.lang.Exception  
      └─ java.io.IOException
```

i.e. checked exception

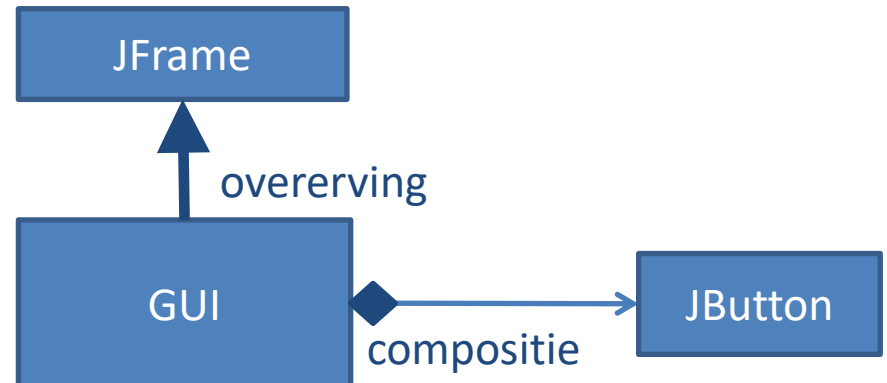
Grafische toepassing

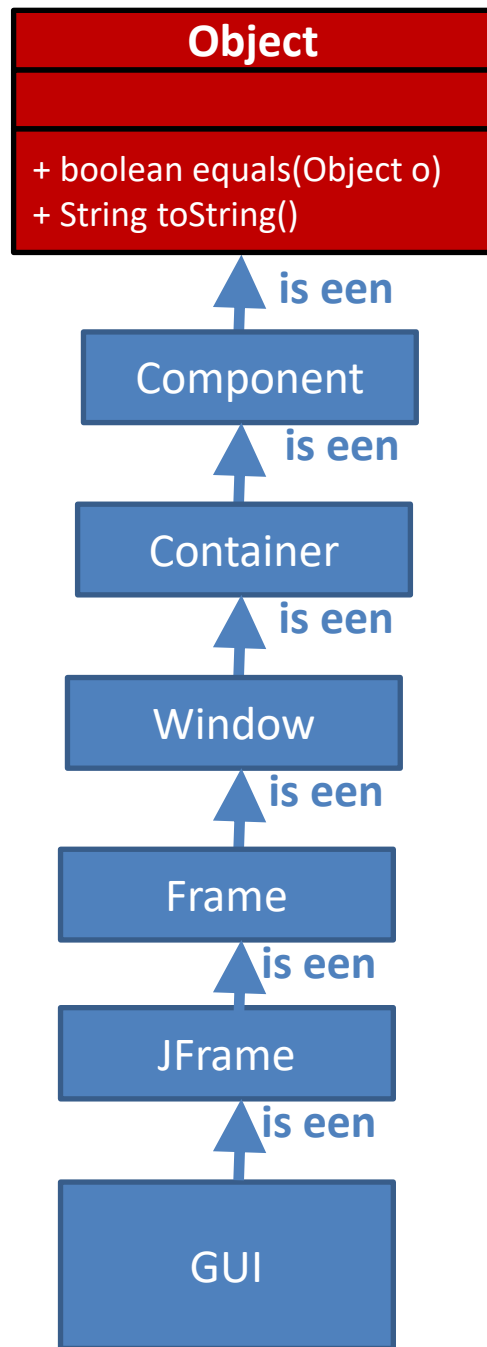


```
public class GUI extends javax.swing.JFrame {  
  
    // Variables declaration - do not modify  
    private javax.swing.JButton jButton;  
    // End of variables declaration  
}
```

java.lang.Object
└─ java.awt.Component
 └─ java.awt.Container
 └─ java.awt.Window
 └─ java.awt.Frame
 └─ javax.swing.JFrame

java.lang.Object
└─ java.awt.Component
 └─ java.awt.Container
 └─ javax.swing.JComponent
 └─ javax.swing.AbstractButton
 └─ javax.swing.JButton





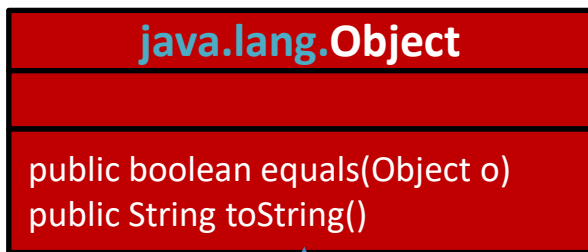
Grafische toepassing

```
public class Object {  
    public String toString() { ...  
}  
  
    public boolean equals(Object o) { ...  
}
```

...

```
public class JFrame extends Frame {  
    ...  
}
```

```
public class GUI extends JFrame {  
    ...  
}
```



```
package java.lang;
```

```
public class Object {  
    ...  
}
```

...

```
package javax.swing;
```

```
import java.awt.*;
```

```
public class
```

...

```
}
```

```
package javax.swing;
```

```
import java.awt.Frame;
```

```
public class JFrame extends Frame {
```

...

```
}
```

```
package presentatie;
```

```
import javax.swing.*;
```

```
public class
```

...

```
}
```

```
package presentatie;
```

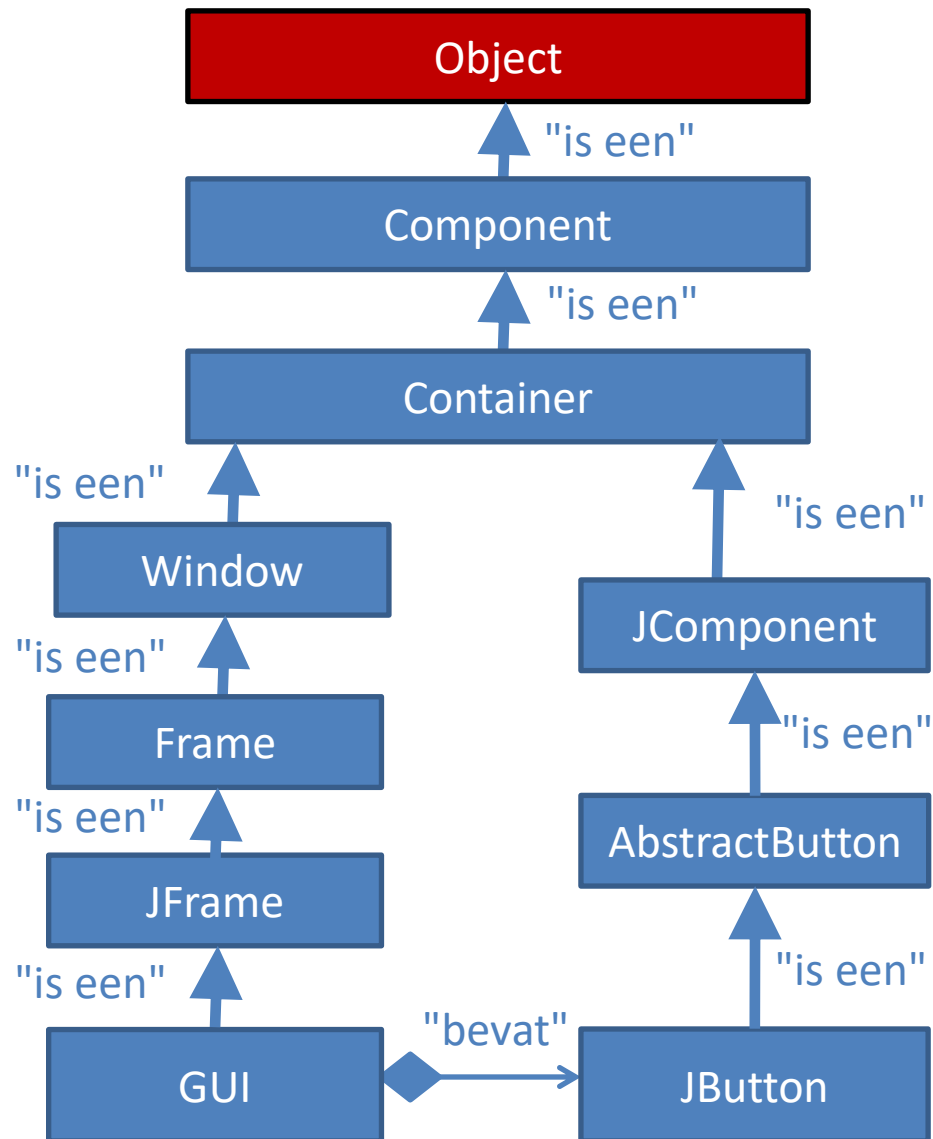
```
import javax.swing.JFrame;
```

```
public class GUI extends javax.swing.JFrame {
```

...

```
}
```

GUI-venster & knop (UML)



GUI-venster & knop (Code)

```
import javax.swing.JFrame;  
import javax.swing.JButton;  
  
public class GUI extends JFrame {  
    private JButton jButton;  
    ...  
}
```


class X extends Y

"To extend" betekent letterlijk "uitbreiden":

In de subklasse kan je de set van velden en methoden uit de basisklasse "uitbreiden" met extra velden en methoden.

Je kan ook bestaande methoden een andere invulling (i.e. andere implementatie) geven

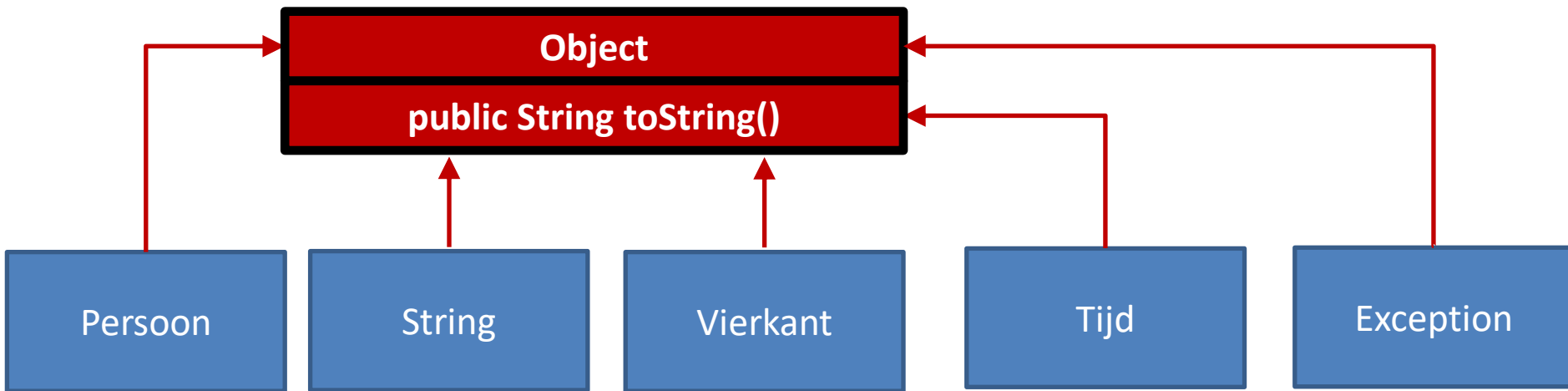
de 'root' methoden (en hun vaste signatuur!)



herimplementatie

- toString **public String toString()**
- equals

Alle klassen erven de 'root' toString() implementatie...



```
Persoon p = new Persoon();  
System.out.println(p.toString());
```

➔ `Persoon@23a59e`

```
Vierkant v = new Vierkant(..., ...);  
System.out.println(v);
```

➔ `Vierkant@f4ea7e`

toString-(her)implementatie

- **'root' implementatie**

- Cf. root-klasse **Object**

```
Persoon p = new Persoon();  
System.out.println(p);
```

geeft op scherm: `Persoon@12fd46`

- **Herimplementatie door Java-API klasse**

- Bv. klasse **String**

```
String s = "Mijn tekstje";  
System.out.println(s);
```

geeft op scherm: `Mijn tekstje`

- **Herimplementeer in eigen klasse**

- Bv. klasse **Tijd**

```
Tijd t = new Tijd(11, 55);  
System.out.println(t);
```

bvb. op scherm: `11:55`

Herimplementatie van de toString() methode

```
@Override
```

```
public String toString() {
```

```
    ....
```

```
}
```

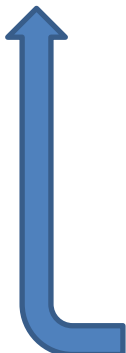
Voorbeeld: Tijd

→ Eerder: Druk zinvolle informatie over object via getter-methoden:

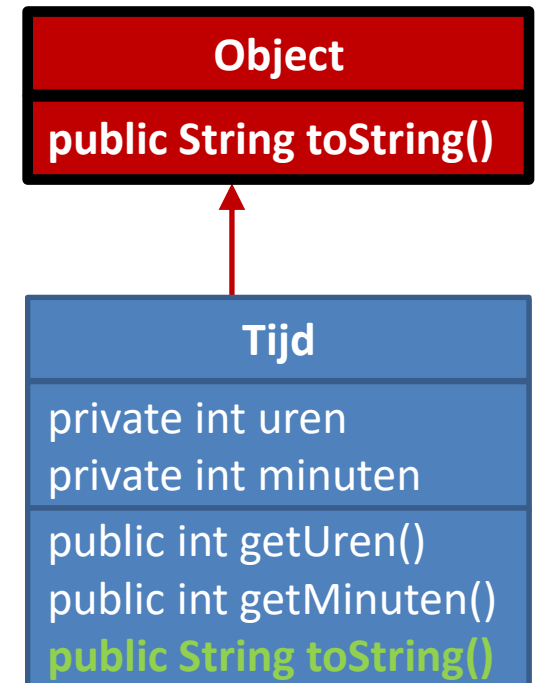
```
//Console.java  
Tijd t = new Tijd(11, 55);  
System.out.println(t.getUren() + ":" + t.getMinuten());
```

→ Beter: Herimplementeer hiervoor de toString() methode:

```
//Console.java  
Tijd t = new Tijd(11, 55);  
System.out.println(t);
```



```
public class Tijd {  
    private int uren;  
    private int minuten;  
  
    @Override  
    public String toString() {  
        return this.uren + ":" + this.minuten;  
    }  
}
```



Voorbeeld: Datum

Datum

```
private int dag  
private int maand  
private int jaar
```

```
public int Datum(int d, int m, int j)  
public String toString()
```

//Console.java

```
Datum d = new Datum(23, 3, 2012);  
System.out.println(d);
```

Gewenste uitvoer:

23/3/'12

```
public class Datum {  
    private int dag, maand, jaar;  
  
    @Override  
    public String toString() {  
        String s = this.dag + "/" + this.maand + "/" + (this.jaar%100);  
        return s;  
    }  
}
```

Voorbeeld: Student

```
public class Console {  
    public static void main(String[] args) {  
        Student[] studenten = new Student[] {  
            new Student("Kristien", 48),  
            new Student("Katja", 42),  
            new Student("Peter", 40)  
        };  
  
        for (Student s : studenten) {  
            System.out.println(s);  
        }  
    }  
}
```

```
1: Kristien (48)  
2: Katja (42)  
3: Peter (40)
```

```
public class Student {  
    static int teller;  
  
    private String naam;  
    private int leeftijd;  
    private int studNr;  
  
    public Student(String naam, int leeftijd) {  
        this.naam = naam;  
        this.leeftijd = leeftijd;  
        this.studNr = ++teller;  
    }  
  
    @Override  
    public String toString() {  
        return studNr + ": " + naam + " (" + leeftijd + ")";  
    }  
}
```


Voorbeeld: Licht & Auto

13 auto's staan stil voor een rood licht.

Telkens wanneer het licht op groen springt, kunnen er 5 auto's doorrijden

```
run:
GROEN LICHT
- BMW 1-AXZ-231 rijdt weg
- MAZDA 1-DGH-739 rijdt weg
- VOLVO 1-HHJD-582 rijdt weg
- CITROEN 1-KXD-398 rijdt weg
- DACIA 1-LEZ-945 rijdt weg
ROOD LICHT
GROEN LICHT
- AUDI 1-ARP-335 rijdt weg
- VOLKSWAGEN 1-BVF-768 rijdt weg
- TOYOTA 1-CGD-703 rijdt weg
- HONDA 1-IUY-218 rijdt weg
- VOLVO 1-EEZ-987 rijdt weg
ROOD LICHT
GROEN LICHT
- KIA 1-DFC-349 rijdt weg
- RENAULT 1-BGK-285 rijdt weg
- FORD 1-ERF-396 rijdt weg
ROOD LICHT
GROEN LICHT
ROOD LICHT
GROEN LICHT
ROOD LICHT
```

//eigenlijk testscenario

```
int cur = 0;
while (true) {
    licht.setKleur(Verkeerslicht.Kleur.groen);
    System.out.println(licht); //eig. licht.toString()
    for (int i = 0; i < 5; i++) {
        if (cur < autos.length) {
            autos[cur].rij();
            System.out.println("- " + autos[i] + " rijdt weg");
            //eig. autos[i].toString()

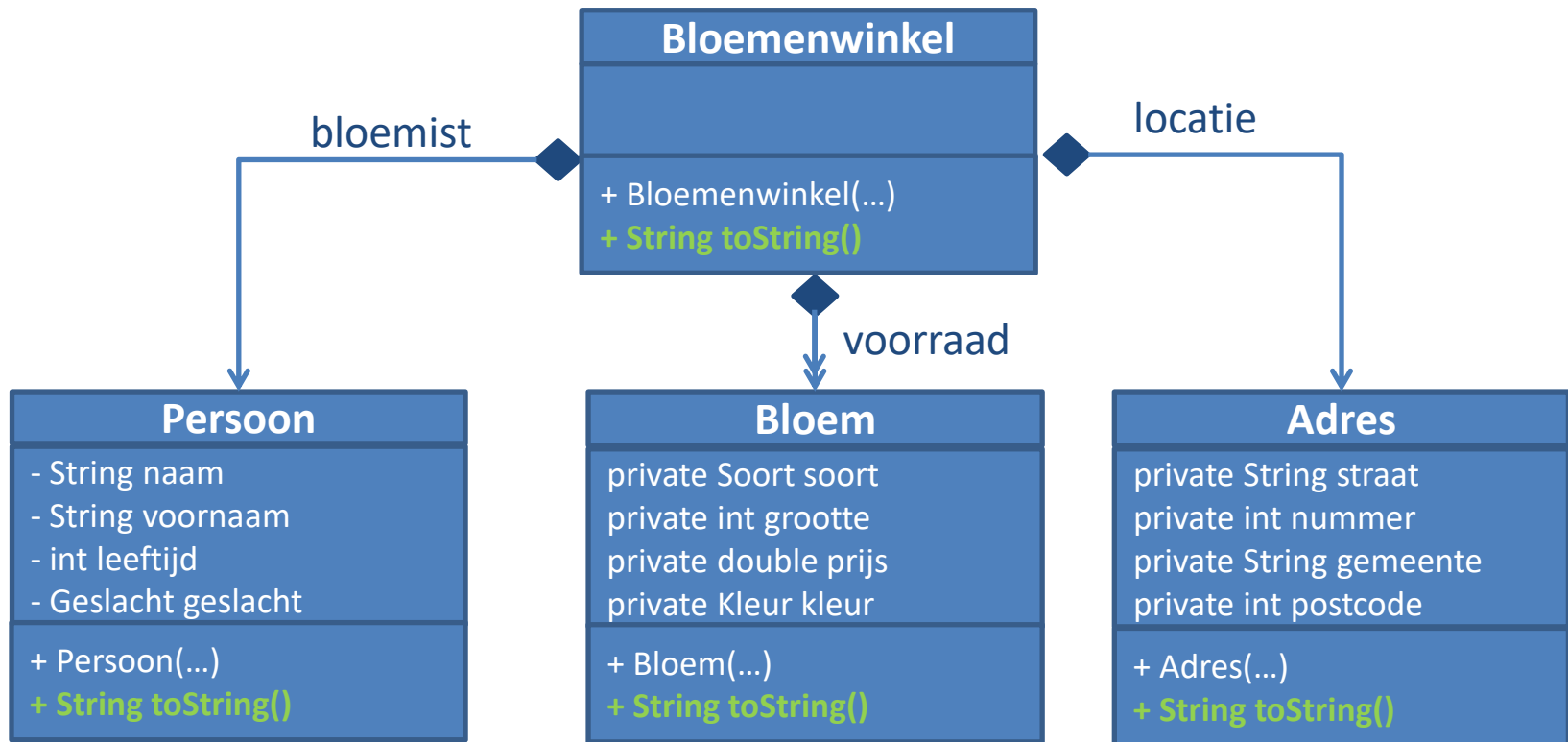
            cur++;
        }
    }
    licht.setKleur(Verkeerslicht.Kleur.rood);
    System.out.println(licht); //eig. licht.toString()

    new Scanner(System.in).nextLine();
}
```

```
public class Auto {  
    private Merk merk;  
    private String nummerplaat;  
  
    public Auto(Merk merk) {  
        this.merk = merk;  
        this.nummerplaat = genereerNummerplaat();  
    }  
  
    private String genereerNummerplaat() {  
        ...  
    }  
  
    public void rij() {  
        this.kmStand += 1;  
    }  
  
    @Override  
    public String toString() {  
        return this.merk + " " + this.nummerplaat;  
    }  
}
```

```
public class Verkeerslicht {  
    public enum Kleur {  
        rood, oranje, groen  
    }  
  
    private Kleur kleur;  
  
    public void setKleur(Kleur kleur) {  
        this.kleur = kleur;  
    }  
  
    public Kleur getKleur() {  
        return this.kleur;  
    }  
  
    @Override  
    public String toString() {  
        return kleur.toString().toUpperCase()  
            + " LICHT";  
    }  
}
```

Voorbeeld: Bloemenwinkel toString() en compositie



Voorbeeld: Bloemenwinkel

```
//Console.java
```

```
Persoon p = new Persoon(...);  
Bloem[] rijVanBloemen = new Bloem[] {...};  
Adres a = new Adres(...);  
Bloemenwinkel winkel = new Bloemenwinkel(p, rijVanBloemen, a);  
System.out.println(winkel);
```

Bloemenwinkel

- Persoon bloemist
- Bloem[] voorraad
- Adres locatie

+ Bloemenwinkel(...)
+ String toString()

‘Overschrijf’ in klasse Bloemenwinkel de toString() methode. Vervang m.a.w. de default implementatie van de root klasse Object door een eigen, betekenisvolle invulling

Stappen:

- Roep toString() voor dataveld bloemist
>> klasse **Persoon**
- Roep toString() voor dataveld locatie
>> klasse **Adres**
- Roep toString() voor ‘elke’ bloem in voorraad
(for-lus >> klasse **Bloem**)

Bloemenwinkel@62937c

Eigenaar: Mevr. Florimont Rosa
Adres: Bloemenweg 17, 9000 GENT
Bloemengamma:

- narcis GEEL 12 cm (1.0 euro)
- narcis GEEL 12 cm (1.0 euro)
- narcis GEEL 12 cm (1.0 euro)
- tulp ZWART 12 cm (1.5 euro)
- tulp WIT 12 cm (1.5 euro)
- roos ROOD 12 cm (2.0 euro)

toString() in klasse Persoon

```
public class Persoon {
```

```
....
```

```
@Override
```

```
public String toString() {
```

```
    String s = "";
```

```
    if (this.isVolwassen() && this.geslacht == Geslacht.MAN) {
```

```
        s += "Mr. ";
```

```
    }
```

```
    else if (this.isVolwassen() && this.geslacht == Geslacht.VROUW) {
```

```
        s += "Mevr. ";
```

```
    }
```

```
    return s += this.naam + " " + this.voornaam;
```

```
}
```

```
}
```

Persoon

- String naam
- String voornaam
- int leeftijd
- Geslacht geslacht

- + Persoon(...)
- + **String toString()**

//Console.java

```
Persoon p1 = new Persoon("Verhaegen", "Rob", 28, Geslacht.MAN);  
System.out.println(p1);
```

Mr. Verhaegen Rob

```
Persoon p2 = new Persoon("Vogelaere", "Marijke", 32, Geslacht.VROUW);  
System.out.println(p2);
```

Mevr. Vogelaere Marijke

```
Persoon p3 = new Persoon("Verhaegen", "Stefke", 8, Geslacht.MAN);  
System.out.println(p3);
```

Verhaegen Stefke

toString() in klasse Adres

Gewenste uitvoer:

`Boswegel 3, 9000 GENT`

```
public class Adres {  
    ...  
    @Override  
    public String toString() {  
        String s;  
        s = this.straat + " " + this.nummer + ", ";  
        s += this.postcode + " " + this.gemeente.toUpperCase();  
  
        return s;  
    }  
}
```

Adres

- String straat
- int nummer
- String gemeente
- int postcode

```
public Adres(...)  
public String toString()
```

toString() in klasse Bloem

```
public class Bloem {  
    ....  
  
    @Override  
    public String toString() {  
        return this.soort + " " + this.kleur + " "  
            + this.lengte + " cm (" + this.prijs + " euro)";  
    }  
}
```

Bloem

private Soort soort
private int grootte
private double prijs
private Kleur kleur

+ Bloem(...)

+ **String toString()**

toString() in klasse Bloemenwinkel

```
public class Bloemenwinkel {  
    private Persoon bloemist;  
    private Bloem[] voorraad;  
    private Adres locatie;  
  
    @Override  
    public String toString() {  
        String info = "Eigenaar:" + this.bloemist + "\n";  
        info += "Adres" + this.locatie + "\n";  
        info += "Bloemengamma: \n";  
  
        for (Bloem bloem : this.voorraad) {  
            if (bloem != null) {  
                info += "\t - " + bloem + "\n";  
            }  
        }  
  
        return info;  
    }  
}
```

```
Eigenaar: Mevr. Florimont Rosa  
Adres: Bloemenweg 17, 9000 GENT  
Bloemengamma:  
    - narcis GEEL 12 cm (1.0 euro)  
    - narcis GEEL 12 cm (1.0 euro)  
    - narcis GEEL 12 cm (1.0 euro)  
    - tulp ZWART 12 cm (1.5 euro)  
    - tulp WIT 12 cm (1.5 euro)  
    - roos ROOD 12 cm (2.0 euro)
```

de 'root' methoden (en hun vaste signatuur!)

- toString

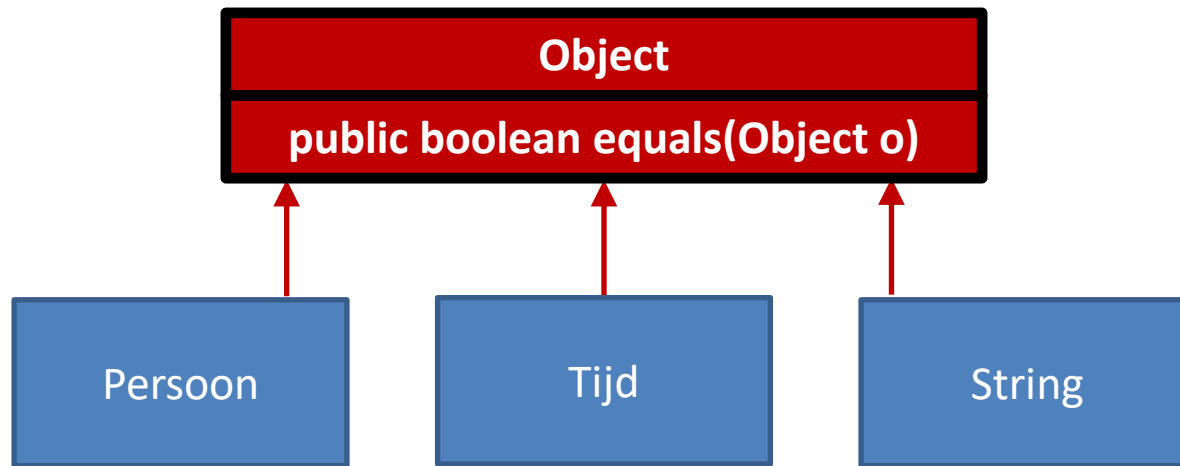
- equals

public boolean equals (Object o)



herimplementatie

Alle klassen erven de 'root' equals() functionaliteit



```
Persoon p1 = new Persoon(...);
Persoon p2 = new Persoon(...);
```

```
if (p1.equals(p2)) {
    ....
}
```

false

```
Tijd t1 = new Tijd(...);
Tijd t2 = t1;
```

```
if (t1.equals(t2)) {
    ....
}
```

true

```
String s1 = new String("test");
String s2 = new String("test");
```

```
if (s1.equals(s2)) {
    ....
}
```

true

equals-(her)implementatie

- Herimplementatie door Java-API klasse

- Bv. klasse String

```
String s1 = new String("aap");  
String s2 = new String("aap");  
  
if (s1.equals(s2)) {  
    ...  
}
```

true

```
String s1 = new String("appel");  
String s2 = new String("ananas");  
  
if (s1.equals(s2)) {  
    ...  
}
```

false

- Herimplementatie in eigen klasse

- Bv. klasse Tijd

```
Tijd t1 = new Tijd(3,25);  
Tijd t2 = new Tijd(7,14);  
  
if (t1.equals(t2)) {  
    ...  
}
```

<Wanneer zijn voor jou
2 tijden gelijk aan elkaar ? >

Herimplementatie van de equals() methode

```
@Override
```

```
public boolean equals(Object o) {  
    ...  
}
```

Voorbeeld: Tijd

```
Tijd t1 = new Tijd(3, 25);  
Tijd t2 = new Tijd(7, 14);
```

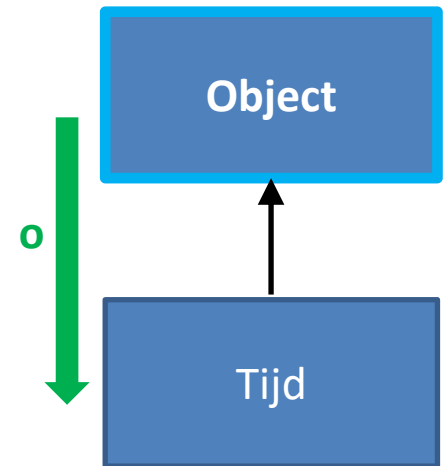
```
if (t1.equals(t2)) {  
    ...  
}
```

```
Tijd nu = new Tijd(10, 00);  
Tijd alarm = new Tijd(12, 00);
```

```
while (!nu.equals(alarm)) {  
    nu.telTijdBij(new Tijd(0, 5));  
}
```

Zie verder
voor een
concreet
codevoorbeeld

```
public class Tijd {  
    private int uren;  
    private int minuten;  
  
    @Override  
    public boolean equals(Object o) {  
        Tijd t = (Tijd)o; //expliciete downcast  
        return this.uren == t.uren && this.minuten == t.minuten;  
    }  
}
```



Let op voor `NullPointerException` & `ClassCastException` !!

Tijd
private int uren private int minuten
public Tijd(int u, int m) public String toString() public boolean equals(Object o)

```
Tijd t1 = new Tijd(...);
Tijd t2;
```

```
if (t1.equals(t2)) {
    ...
}
```

```
Tijd t = new Tijd(...);
Persoon p = new Persoon(...);
```

```
if (t.equals(p)) {
    ...
}
```

```
public class Tijd {
    private int uren;
    private int minuten;

    @Override
    public boolean equals(Object o) {
        if (o == null || !(o instanceof Tijd)) {
            return false;
        }

        Tijd t = (Tijd)o; //expliciete downcast

        return this.uren == t.uren && this.minuten == t.minuten;
    }
}
```

Datum
<ul style="list-style-type: none"> - int dag - int maand - int jaar
<ul style="list-style-type: none"> + Datum(int d, int m, int j) + String toString() + boolean equals(Object o)

```
Datum d1 = new Datum(18, 3, 2015);
Datum d2 = new Datum(1, 4, 2015);
```

```
if (d1.equals(d2)) {
    ...
}
```

```
public class Datum {
    ...
    @Override
    public boolean equals(Object o) {
        if (o == null || !(o instanceof Datum)) {
            return false;
        }

        Datum d = (Datum)o; //expliciete downcast
        return (this.dag == d.dag) && (this.maand == d.maand) && (this.jaar == d.jaar);
    }
}
```


Adres

```
private String straat  
private int nummer  
private String gemeente  
private int postcode
```

```
public Adres(String straat, int nr, int postcode, String gemeente)  
public String toString()  
public boolean equals(Object o)
```

```
Adres a1 = new Adres(...);  
Adres a2 = new Adres(...);
```

```
if (a1.equals(a2)) {  
    ...  
}
```

```
public class Adres {  
    ...  
    @Override  
    public boolean equals(Object o) {  
        if (o == null || !(o instanceof Adres)) {  
            return false;  
        }  
  
        Adres a = (Adres)o; //expliciete downcast  
        return this.straat.equals(a.straat)  
            && this.nummer == a.nummer  
            && this.postcode == a.postcode;  
    }  
}
```

EXTRA OEFENINGEN

op equals & toString

- Auto
- Persoon
- Tijd
- Rapport
- Breuk

Oefening1 - Auto

Auto

```
private Merk merk  
private double prijs  
private Kleur kleur  
private int kmStand
```

```
public Auto()  
public Auto(Merk merk, Kleur kleur,  
            double nieuwPrijs)  
public double getPrijs()  
public Kleur getKleur()  
public void setPrijs(double prijs)  
public void setKleur(Kleur kleur)  
public String toString()  
public boolean equals(Object o)
```

```
//Console.java
```

```
Auto a = new Auto(Merk.AUDI, Kleur.WIT, 30000);  
System.out.println(a);
```

Gewenste uitvoer: **AUDI wit 30000.0 euro (1000 km)**

```
import logica.Kleur;
```

```
public class Auto {  
    @Override  
    public String toString() {  
        return this.merk + " "  
            + this.kleur.toString().toLowerCase() + " "  
            + this.prijs + " euro"  
            + " (" + this.kmStand + " km)";  
    }  
}
```

Beschouw 2 auto's als gelijk als ze dezelfde waarden hebben voor alle velden van de klasse

Auto
private Merk merk private double prijs private Kleur kleur private int kmStand
public Auto() public Auto(String merk, double prijs, Kleur kleur, int kmStand) public double getPrijs() public Kleur getKleur() public void setPrijs(double prijs) public void setKleur(Kleur kleur) public String toString() public boolean equals(Object o)

```
public class Auto {  
    ...  
  
    @Override  
    public boolean equals(Object o) {  
        if (o == null && !(o instanceof Auto)) {  
            return false;  
        }  
  
        Auto a = (Auto)o;  
  
        return this.merk == a.merk  
            && this.kleur == a.kleur  
            && this.prijs == a.prijs  
            && this.kmStand == a.kmStand;  
    }  
}
```

Oefening2 - Persoon

Persoon
<pre>private String naam private String voornaam private int leeftijd private Geslacht geslacht</pre>
<pre>public Persoon(String n, String v, int l, Geslacht g) public String toString() public boolean equals(Object o)</pre>

equals() herimplementatie

Persoon

```
private String naam  
private String voornaam  
private int leeftijd  
private Geslacht geslacht
```

```
public Persoon(String n, String v, int l,  
public String toString()  
public boolean equals(Object o)
```

```
public class Persoon {  
    ...  
    @Override  
    public boolean equals(Object o) {  
        if (o == null || !(o instanceof Persoon)) {  
            return false;  
        }  
  
        Persoon p = (Persoon)o; //expliciete downcast  
  
        return this.naam.equalsIgnoreCase(p.naam)  
            && this.voornaam.equalsIgnoreCase(p.voornaam)  
            && this.leeftijd == p.leeftijd  
            && this.geslacht == p.geslacht;  
    }  
}
```

Opmerking

```
Persoon p = (Persoon)o;  
this.naam.equalsIgnoreCase(p.naam);
```

Zonder tussenvariabele p

NOK: `this.naam.equalsIgnoreCase((Persoon)o.naam)`

Ok: `this.naam.equalsIgnoreCase(((Persoon)o).naam)`

Vorrangsregel:

eerst memberoperator (.),
dan pas cast-operator

```
public class Persoon {  
    ...  
    @Override  
    public boolean equals(Object o) {  
        if (o == null || !(o instanceof Persoon)) {  
            return false;  
        }  
  
        return this.naam.equalsIgnoreCase( ( (Persoon)o ).naam )  
            && this.voornaam.equalsIgnoreCase( ( (Persoon)o ).voornaam )  
            && this.leeftijd == ( (Persoon)o ).leeftijd;  
    }  
}
```

Oefening3 - Tijd

```
import logica.Tijd;

public class Lestijd {
    public static void main(String[] args) {
        Tijd t1 = new Tijd(9, 45);
        Tijd t2 = new Tijd(11, 10);

        System.out.println(t1);
        System.out.println(t2);
        System.out.println();

        while (!t1.equals(t2)) {
            t1.telTijdBij(new Tijd(0,5));
            System.out.println(t1);
        }

        System.out.println("DE LES IS GEDAAN");
    }
}
```

zonder equals() en zonder toString()

```
logica.Tijd@4aa8f0b4
logica.Tijd@7960847b

logica.Tijd@4aa8f0b4
logica.Tijd@4aa8f0b4
logica.Tijd@4aa8f0b4
...
```



```

import logica.Tijd;

public class Lestijd {
    public static void main(String[] args) {
        Tijd t1 = new Tijd(9, 45);
        Tijd t2 = new Tijd(11, 10);

        while (!t1.equals(t2)) {
            System.out.println(t1);
            t1.telTijdBij(new Tijd(0,5));
        }

        System.out.println(t1 + " - DE LES IS GEDAAN");
    }
}

```

met gepaste equals() en toString()

```

09:45
09:50
09:55
10:00
10:05
10:10
10:15
...
11:05
11:10 - DE LES IS GEDAAN

```

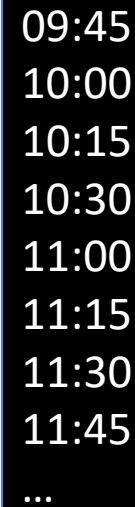
```
import logica.Tijd;

public class Lestijd {
    public static void main(String[] args) {
        Tijd t1 = new Tijd(9, 45);
        Tijd t2 = new Tijd(11, 10);

        while (!t1.equals(t2)) {
            System.out.println(t1);
            t1.telTijdBij(new Tijd(0,15));
        }

        System.out.println(t1 + " - DE LES IS GEDAAN");
    }
}
```

met gepaste equals() en toString()



09:45
10:00
10:15
10:30
11:00
11:15
11:30
11:45
...