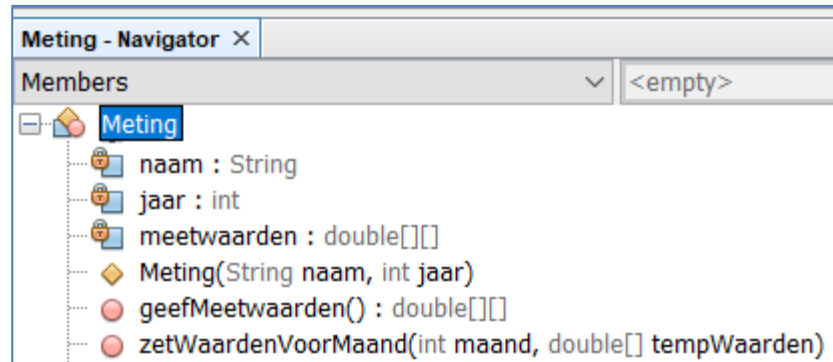


Korte reflectie Labo1: private methoden

```
public static void main(String[] args) {  
    Meting m = new Meting();  
    double[] waarden;  
  
    for (int maand = 0; maand < 12; maand++) {  
        waarden = genereerWaarden(maand); //oproep private methode  
        m.meetwaarden[i] = waarden;  
    }  
  
    drukWaarden(m.meetwaarden); //oproep private methode  
}  
  
private static double[] genereerWaarden(int maand) {  
    ...  
}  
  
private static void drukWaarden(double[] waarden) {  
    ...  
}
```

Reflectie Labo1:

private velden & publieke setters&getters



```
public static void main(String[] args) {  
    Meting m = new Meting();  
    double[] waarden;  
  
    for (int maand = 0; maand < 12; maand++) {  
        waarden = genereerWaarden(maand);  
        m.zetWaardenVoorMaand(maand, waarden); //oproep publieke setter  
    }  
  
    drukWaarden(m.geefMeetwaarden()); //oproep publieke getter  
}
```

Voorbeeld van andere mogelijke interface naar logische klasse

```
public static void main(String[] args) {  
    Meting m = new Meting();  
    double[][] waarden = new double[12][];  
  
    for (int maand = 0; maand < 12; maand++) {  
        waarden[maand] = genereerWaarden(maand);  
    }  
  
    m.zetMeetwaarden(waarden); //2-DIM rij als parameter  
    drukWaarden(m.getMeetwaarden());}
```

Good practice: Initialiseer dataveld in constructor

```
public class Meting {  
    private double[][] meetwaarden;    // declaratie  
  
    public Meting() {  
        meetwaarden = new double[12][]; //initialisatie  
    }  
  
    ...  
}
```

Les 2

Java packages

Opbouw programma: presentatie versus logica

Uniforme geheugenvoorstelling (stack & heap)

Copy constructor

De member operator '.'

Exception handling: throw & try/catch

Samenspel van meerdere logische klassen

Enumeraties

Packages in Java

<https://docs.oracle.com/javase/8/docs/api/>

Package java.lang (default):

- java.lang.String
- java.lang.Math
- java.lang.Integer
- java.lang.Double
- java.lang.System
- ...

Package java.util:

- java.util.Scanner
- ...

Package java.time:

- java.time.LocalDate
- ...

```
import java.util.Scanner;

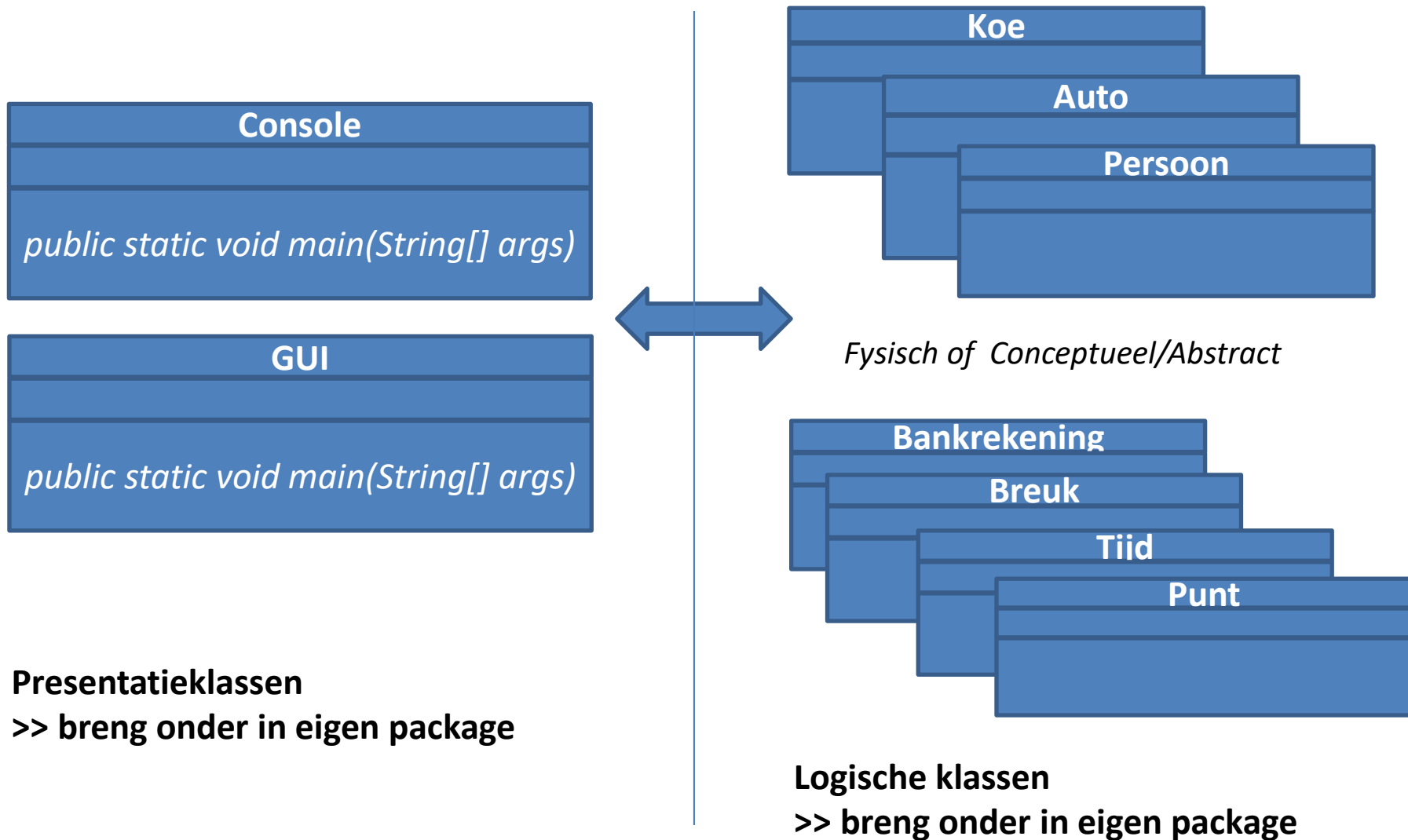
public class BasicIO {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Wat is je naam?");
        String naam = sc.nextLine();
        System.out.println("Dag " + naam);
    }
}
```

```
import java.time.LocalDate;

public class DemoLocalDate {
    public static void main(String[] args) {
        LocalDate today = LocalDate.of(2019, 1, 13);
        System.out.println(today);
    }
}
```

Presentatie versus Logica



packages en import

```
package presentatie;
```

```
import logica.LogischeKlasse;
```

```
public class Presentatieklasse {  
    //doe iets met de geïmporteerde logica  
}
```



```
package logica;
```

```
public class LogischeKlasse {  
    ...  
}
```


Voorbeeld

```
package presentatie;
```

```
import logica.Persoon;
```

```
public class Console {  
    public static void main(String[] args) {  
        //maak object(en) van 'logische' klasse  
        //gebruik zijn logica/functionaliiteit  
    }  
}
```



```
package logica;
```

```
public class Persoon {  
    public Persoon() {...}  
    public Persoon(...,...) {...}  
}
```

Privaat veld met 'gecontroleerde' toegang via getter- en setter-methode 😊

```
public class Persoon {  
    private int leeftijd;  
  
    public int getLeeftijd() {  
        return leeftijd;  
    }  
  
    public void setLeeftijd(int leeftijd) {  
        if (leeftijd >= 0) {  
            this.leeftijd = leeftijd;  
        }  
    }  
}
```

Getter-varianties mogelijk

```
public class Persoon {  
    private String naam;  
    private String voornaam;  
  
    public Persoon(String naam, String voornaam) {  
        this.naam = naam;  
        this.voornaam = voornaam;  
    }  
  
    public String getNaam() {  
        return this.naam;  
    }  
  
    public String getVoornaam() {  
        return voornaam;  
    }  
  
    public String geefVolledigeNaam() {  
        return voornaam + " " + naam;  
    }  
}
```

Aangepast gebruik vanuit Demoklasse

```
Persoon p = new Persoon("Van Assche", "Kristien");  
System.out.println(p.geefVolledigeNaam());  
p.setLeeftijd(-11);  
System.out.println(p.getLeeftijd());
```

VRAAG:

- Wat is de leeftijd van deze persoon ?!
- Hoe kan je aan de oproepende code duidelijk maken dat iets niet gerealiseerd kon worden?

In logische klasse:

```
public class Persoon {  
    private int leeftijd;  
    ...  
  
    public boolean setLeeftijd(int leeftijd) {  
        if (leeftijd >= 0) {  
            this.leeftijd = leeftijd;  
            return true;  
        }  
        return false;  
    }  
}
```

In presentatieklasse:

```
Persoon p = new Persoon();  
...  
  
if (!(p.setLeeftijd(-11) )) {  
    System.out.println("Oei, mislukt...");  
}
```

```
Persoon p = new Persoon("Bevers", "Bas");  
p.setLeeftijd(-11);
```

Je kan er toch voor zorgen dat de oproepende code **AUTOMATISCH** op de hoogte wordt gesteld van een falend request

– Zie exception handling, verder in deze presentatie -

Klassendiagramma



Logische klasse

```
public class Koe {  
    private String kleur;  
    private boolean gevlekt;  
  
    public Koe() {  
    }  
  
    public Koe(String kleur, boolean gevlekt) {  
        this.kleur = kleur;  
        this.gevlekt = gevlekt;  
    }  
}
```

```
    public String getKleur() {  
        return kleur;  
    }  
  
    public boolean getGevlekt() {  
        return gevlekt;  
    }  
  
    public void setKleur(Kleur kleur) {  
        this.kleur = kleur;  
    }  
  
    public void setGevlekt(boolean gevlekt) {  
        this.gevlekt = gevlekt;  
    }  
  
    public void graas() {  
        ....  
    }  
}
```

Ook getters laten extra controle toe

```
public class Koe {  
    //privaat veld  
    private String kleur;  
    ...  
  
    //getter  
    public String getKleur() {  
        if (kleur != null) {  
            return kleur;  
        } else {  
            return "geen kleur ingesteld";  
        }  
    }  
    ...  
}
```


Null referenties vermijden

```
public class Koe {  
    //privaat veld  
    private String kleur;  
  
    //constructor  
    public Koe() {  
        kleur = "onbepaald/default";  
    }  
  
    //getter  
    public String getKleur() {  
        return kleur;  
    }  
    ...  
}
```

Voorbeeld: Student

```
public class Student {  
    private String naam;  
    private int leeftijd;  
  
    public Student(String naam,  
                    int leeftijd) {  
        this.naam = naam;  
        this.leeftijd = leeftijd;  
    }  
  
    public int getLeeftijd() {  
        return leeftijd;  
    }  
  
    public String getNaam() {  
        if (naam != null) {  
            return naam;  
        } else {  
            return "?";  
        }  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Student[] studenten = new Student[] { new Student("Kris", 18),  
                                                new Student("Karel", 19), new Student("Peter", 18)  
        };  
        sorteerOpLeeftijd(studenten);  
        druk(studenten);  
    }  
  
    private static void sorteerOpLeeftijd(Student[] sRij) {  
        for (int i = 0; i < sRij.length; i++) {  
            for (int j = i + 1; j < sRij.length; j++) {  
                if (sRij[i].getLeeftijd() > sRij[j].getLeeftijd()) {  
                    Student temp = sRij[i];  
                    sRij[i] = sRij[j];  
                    sRij[j] = temp;  
                }  
            }  
        }  
    }  
  
    private static void druk(Student[] sRij) {  
        for (Student s : sRij) {  
            System.out.println(s.getNaam() + " (" + s.getLeeftijd() + ")");  
        }  
    }  
}
```

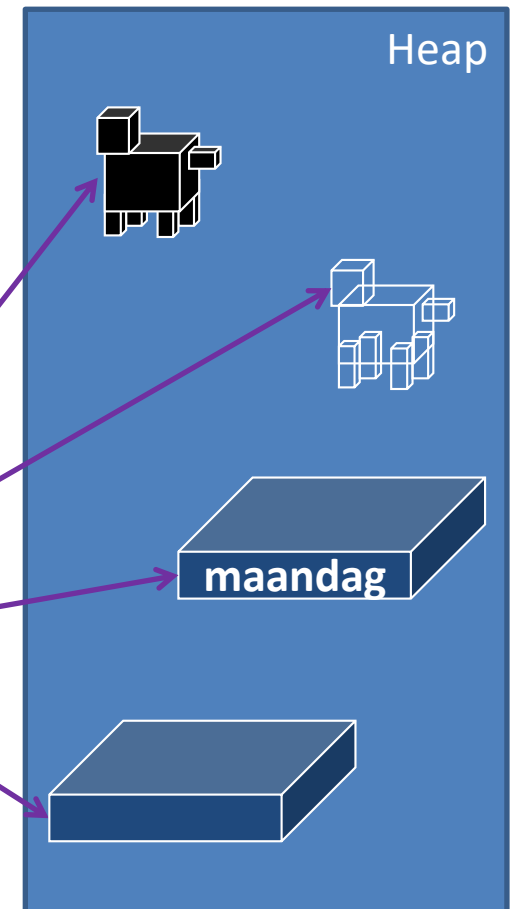
Geheugenallocatie – stack & heap

```
public static void main(String[] args) {  
    String s1 = new String();  
    String s2 = new String("maandag");  
  
    Koe k1 = new Koe();  
    Koe k2 = new Koe("zwart", false);  
}
```

Stack groeit naar
kleinere adressen toe

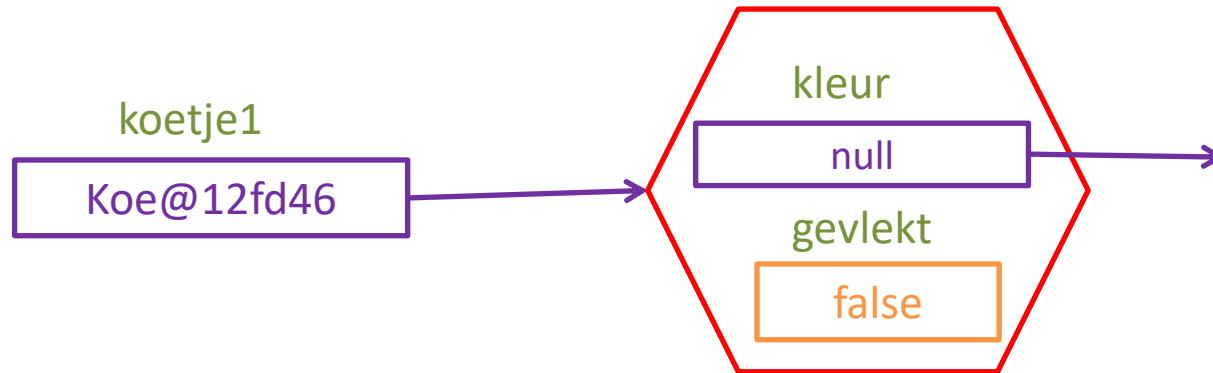
k2	Koe@6d06d69
k1	Koe@15db9742
s2	String@6d06d69
s1	String@15db974

stack

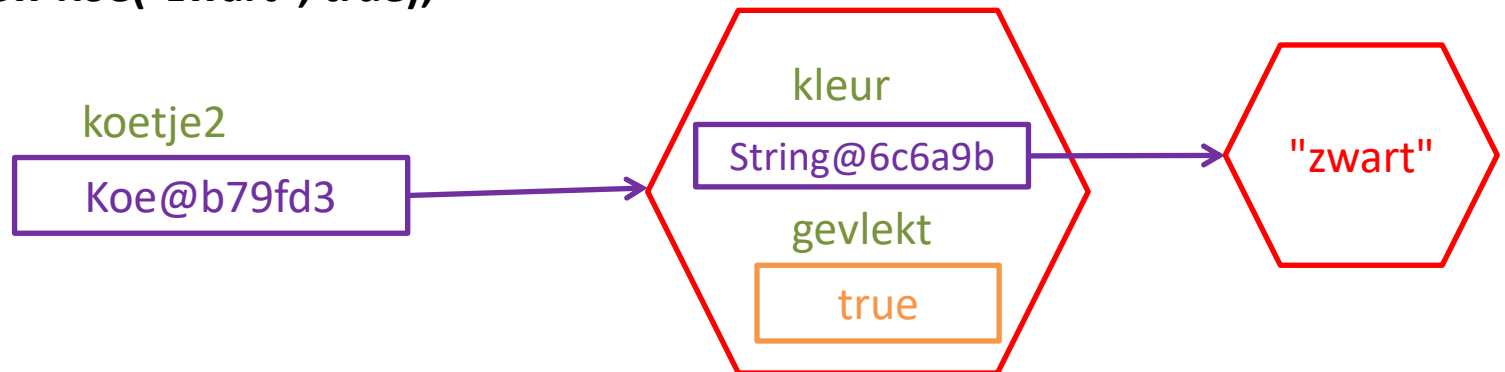


Uniforme geheugenvoorstelling

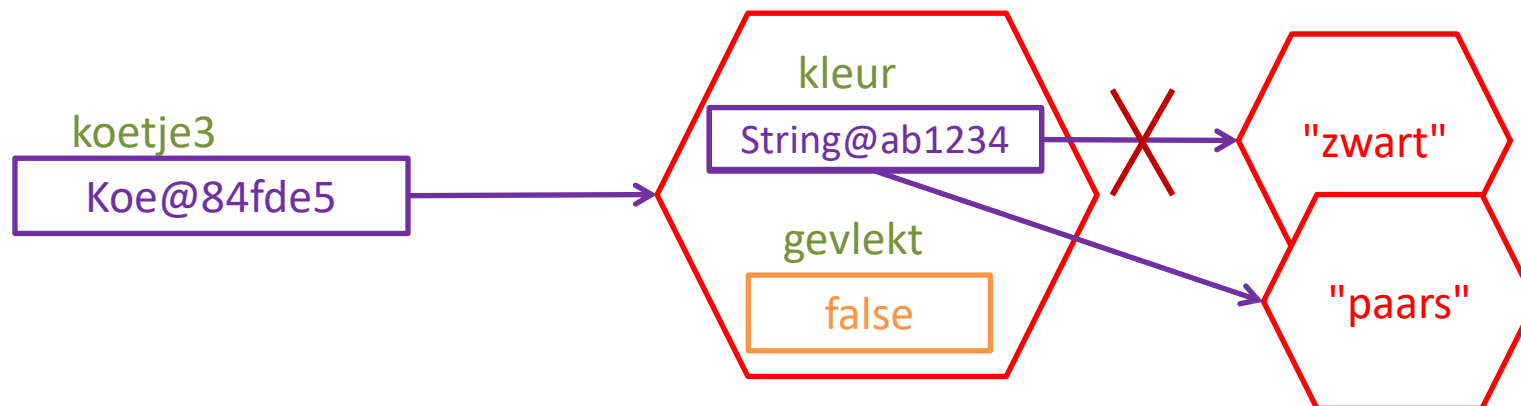
Koe **koetje1** = new Koe();



Koe **koetje2** = new Koe("zwart", true);



```
Koe koetje3 = new Koe("zwart");
```



```
System.out.println(koetje3.getKleur()); //zwart
```

```
koetje3.setKleur("paars");  
of:  
koetje3.setKleur(new String("paars"));
```

In klasse Koe:

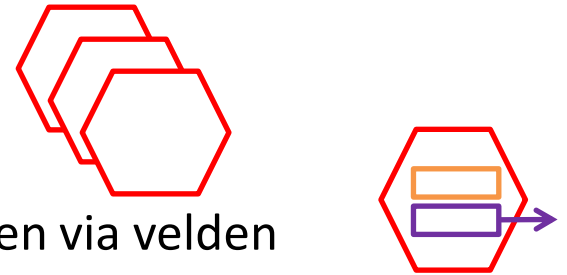
```
public void setKleur(Kleur kleur) {  
    this.kleur = kleur;  
}
```

```
System.out.println(koetje3.getKleur()); //paars
```

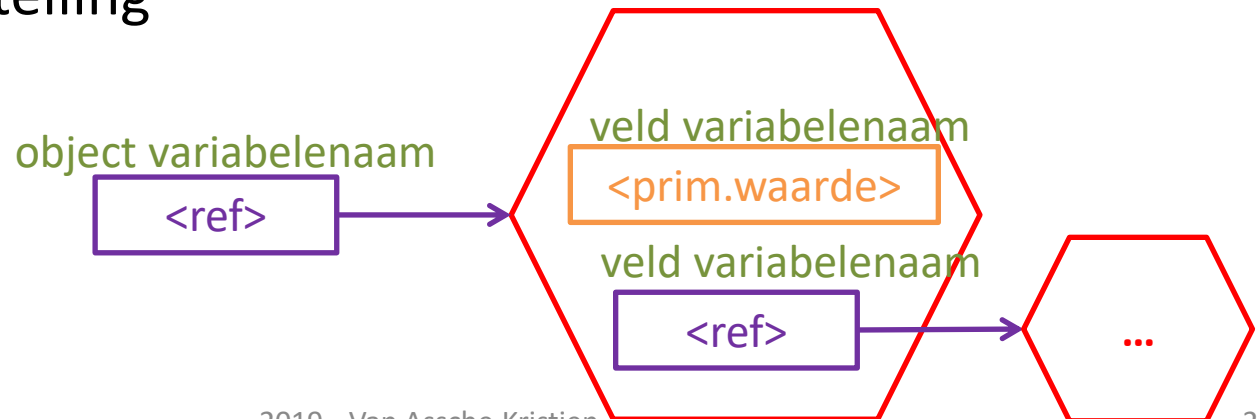
Besluit: Klasse als datatype

- Concepten

- Meerdere objecten maken van eenzelfde klasse
- Diverse soorten 'data' in een klasse onderbrengen via velden
- Elk object krijgt zijn eigen 'data' (bij creatie, evt. nadien aanpasbaar)
- Klasse is een object-type => variabele van dat type is een (object-)referentie
 - veld van het primitieve type => elementaire waarde
 - veld van het object-type => opnieuw referentie (naar object)



- Geheugenvoorstelling



Oefening Boek

- Welke **kenmerken** hou ik bij? Van welk type is de data?
- Welke **constructoren** wil ik ter beschikking stellen?
 - Enkel standaard constructor => autom. gegenereerd
 - Enkel niet-standaard constructor => zelf schrijven
 - Beide constructoren => beide schrijven
 - Meerdere constructoren = constructor overloading
- Welke **functionaliteit** wil ik voorzien? Welke parameters geef ik mee? Wat geef ik terug als resultaat?

Boek

```
public boolean lees(int aantalPg)
```

```
Boek b = new Boek();  
b.lees(12);  
b.lees();
```

Boek

```
private String titel  
private int aantalPaginas
```

```
public Boek(String titel)  
public Boek(String titel, int aantalPg)
```







```
public boolean lees(int aantalPg)
```

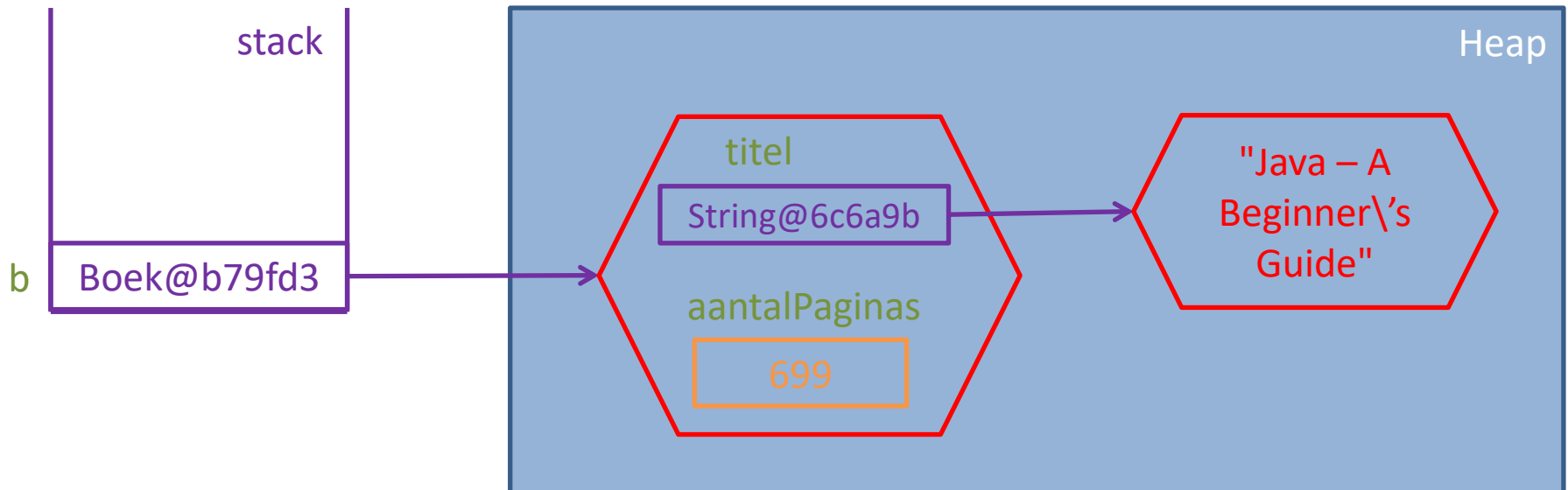
```
Boek b1 = new Boek();  
Boek b2 = new Boek("Beginner's Guide");  
Boek b3 = new Boek("Beginner's Guide", 699);  
Boek b4 = new Boek(699, "Beginner's Guide");  
Boek b5 = new Boek(699);
```

```
do {  
    ...  
} while(b3.lees(10));
```


Netbeans 'Variables' venster

Boek **b** = new Boek("Java – A Beginner\'s Guide", 699);

Name	Type	Value
 b	Boek	 #121
 titel	String	 "Java - A Beginner\'s Guide"
 aantalPaginas	int	 699



Oefening Auto

```
public class Demo {  
    public static void main(String[] args) {  
        Auto a1 = new Auto();  
        Auto a2 = new Auto("BMW");  
        Auto a3 = new Auto("Audi", 30000.0);  
  
        Auto a4 = new Auto("BMW", 20000.0, 12345);  
        a4.rij(100);  
    }  
}
```

Gegeven bovenstaand fragment uit Demo klasse:

- *Schrijf de logische klasse Auto*
- *Wat is de kilometerstand van auto a4 na uitvoer van dit demoprogramma ?*
- *Teken de geheugenvoorstelling van object a4 nadat het codefragment doorlopen is*

```
public class Auto {  
    private String merk;  
    private double prijs;  
    private int kmStand;  
  
    public Auto() {...}  
    public Auto(String merk) {...}  
    public Auto(String merk, double prijs) {...}  
    public Auto(String merk, double prijs, int kmStand) {...}  
  
    public void rij(int km) {  
        this.kmStand += km;  
    }  
}
```

Druk de kilometerstand van auto a4 na het rijden

```
public class Console {  
    public static void main(String[] args) {  
        Auto a1 = new Auto();  
        Auto a2 = new Auto("BMW");  
        Auto a3 = new Auto("Audi", 30000);  
  
        Auto a4 = new Auto("BMW", 20000.0, 12345);  
        a4.rij(100);  
  
        //Getter nodig!!!  
        System.out.println(a4.getKmStand()); //i.e. 12445 km  
    }  
}
```

```

public class Auto {
    private String merk;
    private double prijs;
    private int kmStand;

    public Auto() {...}
    public Auto(String merk) {...}
    public Auto(String merk, double prijs) {...}
    public Auto(String merk, double prijs, int kmStand) {...}

    public int getKmStand() {
        return this.kmStand;
    }

    public void rij(int km) {
        this.kmStand += km;
    }
}

```

Getter:

Niet toestaan dat externen de kilometerstand kunnen terugdraaien

```
public class Console {  
    public static void main(String[] args) {  
        Auto a1 = new Auto();  
        Auto a2 = new Auto("BMW");  
        Auto a3 = new Auto("Audi", 30000);  
  
        Auto a4 = new Auto("BMW", 20000.0, 12345);  
        a4.rij(100);  
  
        int km = a4.getKmStand();  
  
        //niet toestaan dat de kilometerstand gemanipuleerd kan worden  
        a4.setKmStand(km - 1000);  
    }  
}
```

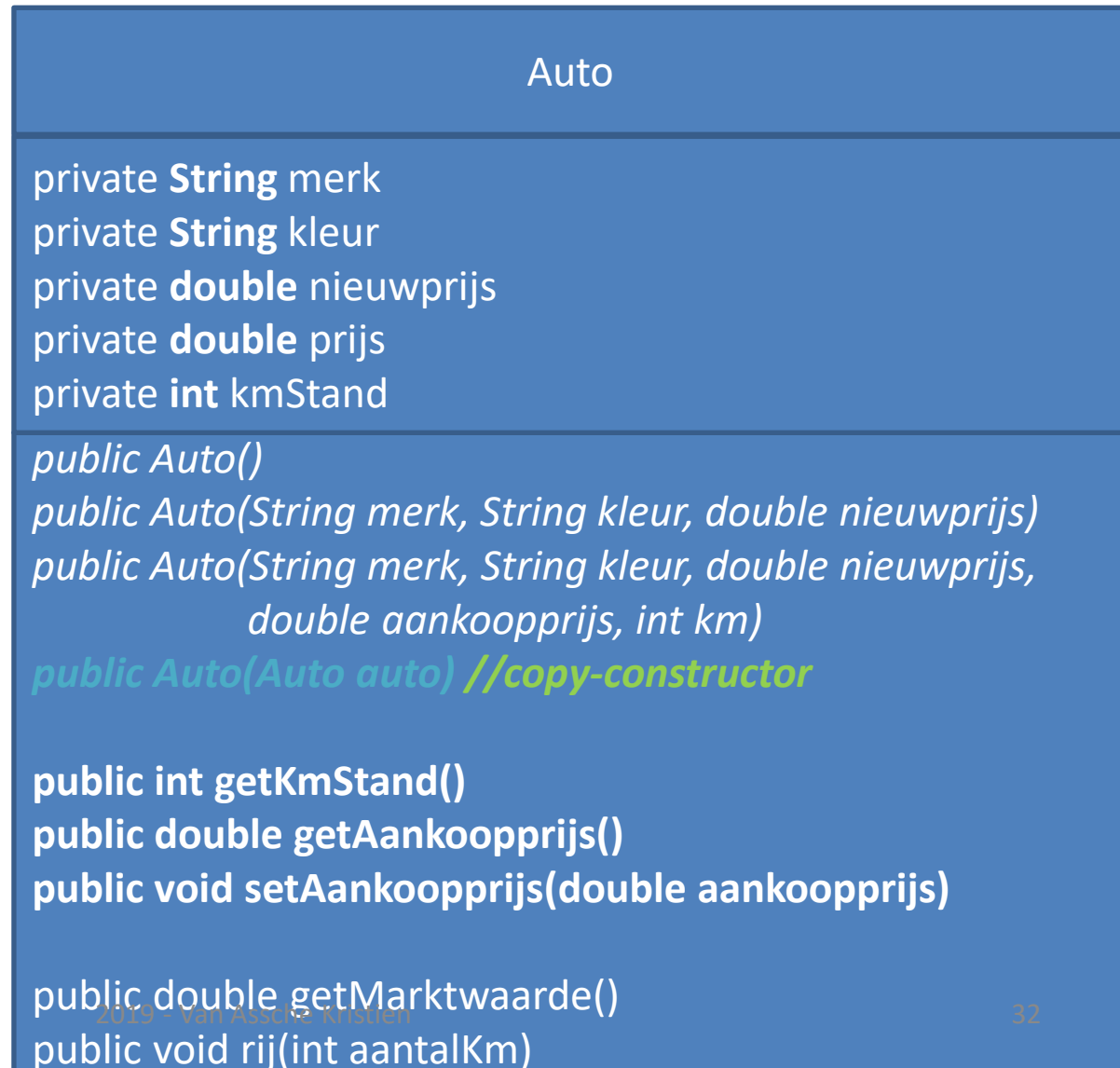
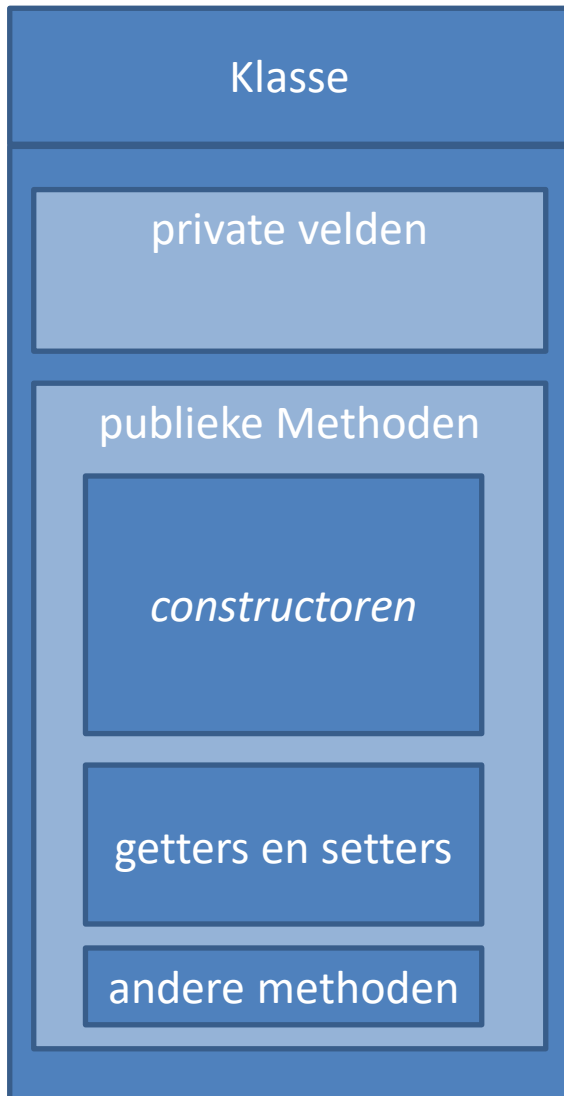
```
public class Auto {  
    private String merk;  
    private double prijs;  
    private int kmStand;  
  
    public Auto() {...}  
    public Auto(String merk) {...}  
    public Auto(String merk, double prijs) {...}  
    public Auto(String merk, double prijs, int kmStand) {...}
```

```
//Voorzie wel een getter  
public int getKmStand() {  
    return this.kmStand;  
}
```

```
//Voorzie géén setter!
```

```
    public void rij(int km) {  
        this.kmStand += km;  
    }  
}
```

Klassendiagramma



Copy-constructor

```
public Auto(Auto a) {  
    merk = a.merk;  
    kleur = a.kleur;  
    this.nieuwprijs = a.nieuwprijs;  
    this.aankoopprijs = a.aankoopprijs;  
    this.kmStand = a.kmStand;  
}
```

MERK OP: Initialisatie van de velden via **<veldnaam>** of **this.<veldnaam>**

Constructor overloading

```
public Auto(String merk, String kleur, double nieuwprijs, double aankoopprijs, int km) {  
    this.merk = merk;  
    this.kleur = kleur;  
    this.nieuwprijs = nieuwprijs;  
    this.aankoopprijs = aankoopprijs;  
    this.kmStand = km;  
}  
  
public Auto(String merk, String kleur, double nieuwprijs) {  
    this(merk, kleur, nieuwprijs, nieuwprijs, 0);  
}
```

MERK OP: 'Interne' oproep van de overloaded constructor via **this(...)**

De member operator '.'

In presentatieklasse (of gelijk welke andere klasse dan de logische klasse Auto):

```
Auto a = new Auto(...);  
a.<publieke_methode>();  
  
public void methode(Auto a) {  
    a.<publieke_methode>();  
}
```

In logische klasse Auto zelf:

```
this.<privaat_veld>;  
this.<publieke_methode>();  
this.<private_methode>();  
  
public void methode(Auto a) {  
    a.<privaat_veld>;  
    a.<publieke_methode>();  
    a.<private_methode>();  
}  
  
public Auto(Auto a) {  
    this.<privaat_veld> = a.<privaat_veld>;  
}
```

```

public class Auto {
    //velden
    private String merk;
    private String kleur;
    private double nieuwprijs;
    private double aankoopprijs;
    private int kmStand;

    //constructoren
    public Auto() {...}
    public Auto(String merk) {...}
    public Auto(String merk, String kleur, double nieuwprijs) {...}
    public Auto(String merk, String kleur, double nieuwprijs, double aankoopprijs, int km) {...}
    public Auto(Auto a)

    //getters & setters
    public int getKmStand() {
        return this.kmStand;
    }

    public double getAankoopprijs() {
        return this.aankoopprijs;
    }

    public void setAankoopprijs(double aankoopprijs) {
        this.aankoopprijs = aankoopprijs;
    }
}

```

```

//andere methoden
public void rij(int km) {
    this.kmStand += km;
}

public void rij() {
    rij(1); //oproep overloaded methode
}

public double getMarktwaaarde() {
    //formule omg. evenredig met kmStand
}

```

Oefening Auto

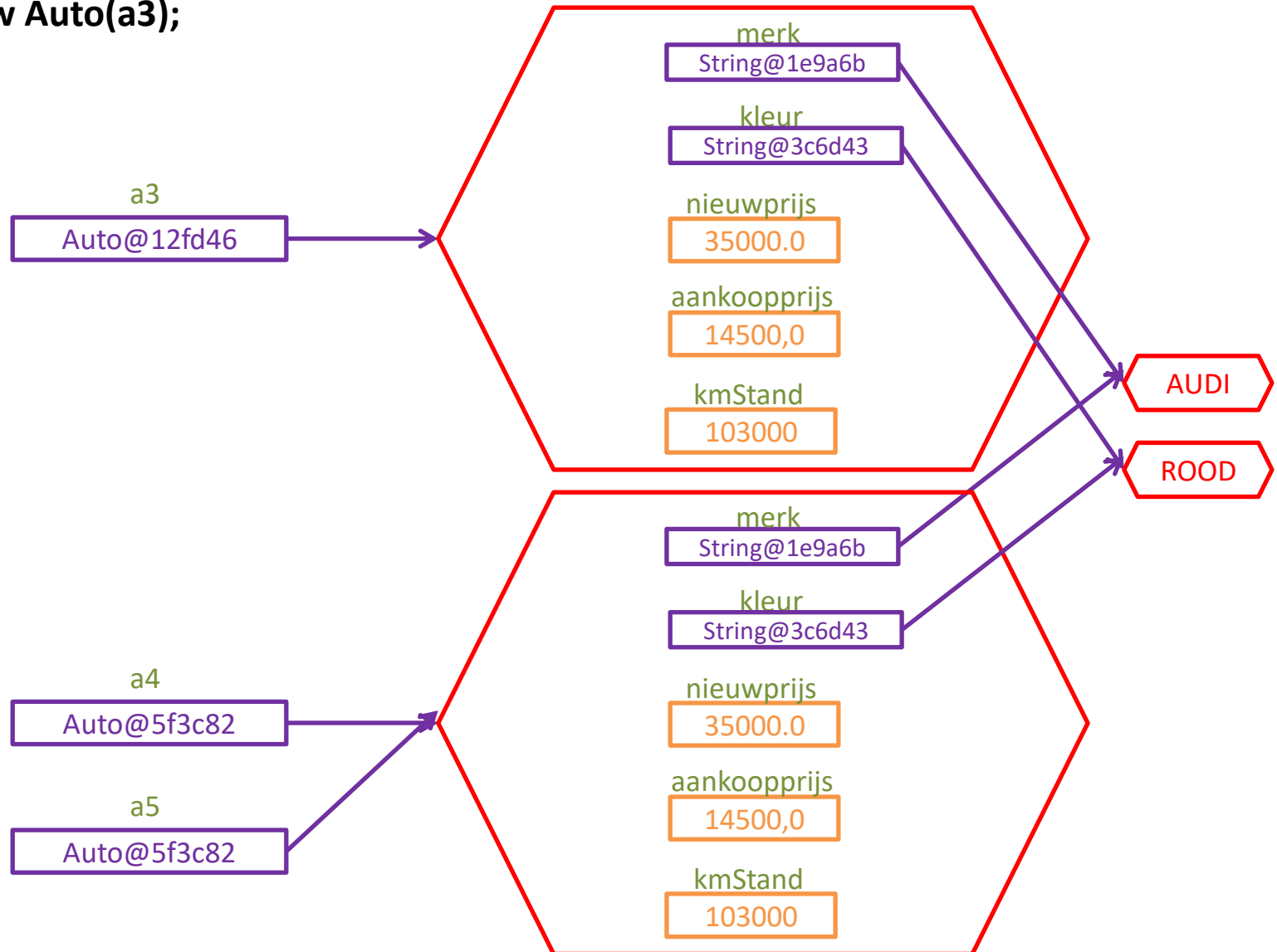
```
public class Console {  
    public static void main(String[] args) {  
        Auto a1 = new Auto();  
        Auto a2 = new Auto("BMW", "WIT", 30000);  
        Auto a3 = new Auto("AUDI", "ROOD", 35000, 14500, 103000);  
        Auto a4 = new Auto(a3);  
        Auto a5 = a4;  
  
        a4.rij(1000);  
  
        //data opvragen  
        int km = a4.getKmStand();  
        double aankoopprijs = a4.getAankoopprijs();  
        double marktwaarde = a4.getMarktwaarde();  
  
        km = a3.getKmStand();  
        km = a5.getKmStand();  
    }  
}
```

Geef voorstelling
in het geheugen

```
//104.000 km  
//14.500€  
// cf. implementatie
```

```
//103.000 km  
//104.000 km
```

```
Auto a3 = new Auto("AUDI", "ROOD", 35000, 14500, 103000);  
Auto a4 = new Auto(a3);  
Auto a5 = a4;
```



Exception Handling

- A. In methode
- B. In constructor

- Algemeen
- Specifiek

A. Exceptie 'opwerpen' (i.e. 'throwen') in **methode** van logische klasse

```
public class Persoon {  
    private int leeftijd;  
  
    public void setLeeftijd(int leeftijd) {  
        if (leeftijd < 0) {  
            throw new IllegalArgumentException("leeftijd mag niet negatief zijn");  
        }  
  
        this.leeftijd = leeftijd;  
    }  
}
```

Resultaat als je een negatieve parameterwaarde meegeeft aan de setter methode:

```
Exception in thread "main" java.lang.IllegalArgumentException: leeftijd mag niet negatief zijn  
    at logica.Persoon.setLeeftijd(Persoon.java:52)  
    at presentatie.Console.main(Main.java:10)
```

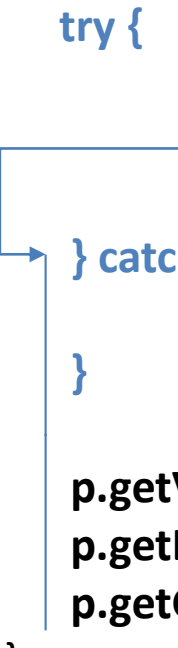

Excepties 'opvangen' (i.e. 'catchen') in presentatieklasse

```
public void testPersoon() {  
    try {  
        Persoon p = new Persoon();  
        p.setNaam("Van Assche");  
        p.setVoornaam("Kristien");  
        p.setLeeftijd(-11);  
        p.setGeslacht(true);  
    } catch (IllegalArgumentException e) {  
        System.out.println("FOUT:" + e.getMessage());  
    }  
  
    System.out.println("Vervolg van programma");  
}
```

FOUT: leeftijd mag niet negatief zijn
Vervolg van programma

Detail

```
public void testPersoon() {  
    Persoon p;  
  
    try {  
        p = new Persoon("Van Assche", "Kristien");  
        p.setLeeftijd(-11);  
        p.setGeslacht(true);  
    } catch (IllegalArgumentException e) {  
        System.out.println("FOUT:" + e.getMessage());  
    }  
  
    p.getVolledigeNaam(); //Kristien Van Assche  
    p.getLeeftijd();      //0  
    p.getGeslacht();      //false  
}
```



De klasse 'IllegalArgumentException'

--constructoren--

```
public class Persoon {  
    private int leeftijd;  
  
    public void setLeeftijd(int leeftijd) {  
        if (leeftijd < 0) {  
            throw new IllegalArgumentException("leeftijd mag niet negatief zijn");  
        }  
  
        this.leeftijd = leeftijd;  
    }  
}
```

java.lang.IllegalArgumentException

private String message

public IllegalArgumentException()
public IllegalArgumentException(String msg)
public String getMessage()

De klasse 'IllegalArgumentException'

--getter methode--

```
public void testPersoon() {  
    try {  
        ...  
    } catch (IllegalArgumentException e) {  
        System.out.println("FOUT:" + e.getMessage());  
    }  
}
```

java.lang.IllegalArgumentException

private String message

```
public IllegalArgumentException()  
public IllegalArgumentException(String msg)  
public String getMessage()
```

FOUT: leeftijd mag niet negatief zijn

Throwen van een 'algemeen' Exception-object

```
public class Persoon {  
    private int leeftijd;  
  
    public void setLeeftijd(int leeftijd) {  
        if (leeftijd < 0) {  
            throw new Exception("leeftijd mag niet negatief zijn");  
        }  
  
        this.leeftijd = leeftijd;  
    }  
}
```

java.lang.Exception

private String message

public Exception()
public Exception(String msg)
public String getMessage()

Opvangen van het algemeen Exception-object 'e'

```
public void testPersoon() {  
    try {  
        ...  
    } catch (Exception e) {  
        System.out.println("FOUT:" + e.getMessage() );  
    }  
}
```

FOUT: leeftijd mag niet negatief zijn

java.lang.Exception

private String message

public Exception()
public Exception(String msg)
public String getMessage()

➔ Hiermee vang je **alle** soorten fouten op, waaronder ook de `IllegalArgumentException`

B. Exceptie 'opwerpen' (i.e. 'throwen') in **constructor** van logische klasse

```
public class Persoon {  
    ...  
    public Persoon(String naam, String voornaam, int leeftijd, Geslacht geslacht) {  
        if (leeftijd < 0) {  
            throw new IllegalArgumentException("leeftijd mag niet negatief zijn");  
        }  
  
        this.naam = naam;  
        this.voornaam = voornaam;  
        this.leeftijd = leeftijd;  
        this.geslacht = geslacht;  
    }  
}
```

1. Wat is de uitvoer van onderstaand testprogramma?

```
public void main() {  
    Persoon p;  
  
    ✗ p = new Persoon("De Paepe", "Tijs", -12, Geslacht.MAN);  
    p.getVolledigeNaam();  
    p.getLeeftijd();  
    p.getGeslacht();  
  
    System.out.println("Test done.");  
}
```

Exception in thread "main" java.lang.IllegalArgumentException: leeftijd mag niet negatief zijn
at presentatie.Console.main(Console.java:4)

2. Wat is de uitvoer van onderstaand programma?


```
public static void main(...) {  
    Persoon p;  
  
    try {  
        p = new Persoon("De Paepe", "Tijs", -12, false);  
    } catch (IllegalArgumentException e) {  
        System.out.println("FOUT: " + e.getMessage() );  
    }  
  
    p.getVolledigeNaam();  
    p.getLeeftijd();  
    p.getGeslacht();  
  
    System.out.println("Test done.");  
}
```

FOUT: leeftijd mag niet negatief zijn

Exception in thread "main" java.lang.NullPointerException
at presentatie.Console.main(Console.java:10)

3. Wat is de uitvoer van onderstaand programma?

```
public static void main(...) {  
    Persoon p;  
  
    try {  
        p = new Persoon("De Paepe", "Tijs", -12, false);  
        p.getVolledigeNaam();  
        p.getLeeftijd();  
        p.getGeslacht();  
    } catch (IllegalArgumentException e) {  
        System.out.println("FOUT: " + e.getMessage() );  
    }  
  
    System.out.println("Test done.");  
}
```



FOUT: leeftijd mag niet negatief zijn

Test done.

De klasse Persoon (vervolledigd)

Navigator Members View

- PersoonOK
- PersoonOK(String naam, String voornaam)
- PersoonOK(String naam, String voornaam, int leeftijd, boolean geslacht)
- PersoonOK()
- getGeslacht() : boolean
- getLeeftijd() : int
- getNaam() : String
- getVolledigeNaam() : String
- getVoornaam() : String
- heeftZelfdeNaamAls(PersoonOK p) : boolean
- isVolwassen() : boolean
- setGeslacht(boolean geslacht)
- setLeeftijd(int leeftijd)
- setNaam(String naam)
- setVoornaam(String voornaam)
- verjaar()
- MAN : boolean
- VROUW : boolean
- geslacht : boolean
- leeftijd : int
- naam : String
- voornaam : String

2 niet-standaard en
1 standaard constructor

5 publieke getter-methoden

4 publieke setter-methoden

2 publieke constanten

4 private velden

Tip: Vermijd zoveel mogelijk null pointer excepties !!

In logische klasse:

```
public void doeletsMetRij(int[] rij) {  
    if (rij != null) {  
        ...;  
    }  
}
```

```
public boolean doeletsMetRij(int[] rij) {  
    if (rij != null) {  
        ...;  
        return true;  
    }  
  
    return false;  
}
```

In presentatie klasse:

```
public static void main(String[] args) {  
    ...  
    obj.doeletsMetRij(rij);  
}
```

```
public static void main(String[] args) {  
    ...  
    if (obj.doeletsMetRij(rij)) {  
        //gelukt  
    }  
}
```

Beperkingen/Bedenkingen

- **Geen controle** of de oproepende code nagaat of de gevraagde functionaliteit effectief is uitgevoerd
 - Opgeworpen exception daarentegen wordt wel opgemerkt, ofwel omdat het programma wordt beëindigd, ofwel omdat de exception via code opgevangen wordt
- **Een constructor heeft geen return type, dus kán geen 'false' teruggeven...**
 - Kan wel zelf een exceptie opwerpen (**throw**)
 - De oproepende code kan die dan opvangen (via **try/catch**-blok)
- **Een methode heeft maar één return type...**
 - Niet altijd mogelijk om (via boolean) aan te geven of er iets fout liep
 - Kan wel een exceptie opwerpen (**throw**)
 - De oproepende code kan die dan opvangen (via **try/catch**-blok)

Voorbeeld: methode faculteit

De faculteit van 6 = $6 * 5 * 4 * 3 * 2 * 1 = 720$

De faculteit van 5 = $5 * 4 * 3 * 2 * 1 = 120$

De faculteit van 4 = $4 * 3 * 2 * 1 = 24$

De faculteit van 3 = $3 * 2 * 1 = 6$

De faculteit van 2 = $2 * 1 = 2$

De faculteit van 1 = 1

De faculteit van 0 = 1

```
public int faculteit(int n) {  
    if (n < 0) {  
        throw new IllegalArgumentException("ongeldige input: " + n + ". Positief getal vereist");  
    }  
  
    if (n == 0) {  
        return 1;  
    }  
  
    int fac = 1;  
    for (int i = 1; i <= n; i++) {  
        fac *= i;  
    }  
  
    return fac;  
}
```

Presentatie klasse:

```
int getal = 6;

while (true) {
    System.out.println("De faculteit van " + getal + ": " + f.faculteit(getal));
    getal--;
}
```

De faculteit van 6: 720

De faculteit van 5: 120

De faculteit van 4: 24

De faculteit van 3: 6

De faculteit van 2: 2

De faculteit van 1: 1

De faculteit van 0: 1

Exception in thread "main" java.lang.IllegalArgumentException: ongeldige input: -1. Positief getal vereist
at logica.Functionaliteiten.faculteit(Functionaliteiten.java:16)
at presentatie.Console.main(Console.java:21)

```

int getal = 6;

while (true) {
    try {
        System.out.println("De faculteit van " + getal + ": " + f.faculteit(getal));
        getal--;
    } catch (IllegalArgumentException e) {
        System.out.println("OPGEVANGEN FOUT: " + e.getMessage());
        break;
    }
}

```

IllegalArgumentException

private String message

public IllegalArgumentException()

public IllegalArgumentException(String msg)

public String getMessage()

⇒ *Constructor stockeert boodschap
in 'message' veld*

IllegalArgumentException

private String message

public IllegalArgumentException()

public IllegalArgumentException(String msg)

public String getMessage()

⇒ *Getter haalt boodschap
uit dat 'message' veld*

De klasse Auto - constructor

```
public Auto(String merk, String kleur, double nieuwprijs, double aankoopprijs, int km) {  
    if (this.nieuwprijs < 0 || aankoopprijs < 0) {  
        throw new IllegalArgumentException("prijzen kunnen niet negatief zijn");  
    }  
    else if (aankoopprijs > nieuwprijs) {  
        throw new IllegalArgumentException(  
            "aankoopprijs kan niet groter zijn dan nieuwwaarde");  
    }  
    else if (km < 0) {  
        throw new IllegalArgumentException("kilometerstand kan niet negatief zijn");  
    }  
  
    this.merk = merk;  
    this.kleur = kleur;  
    this.nieuwprijs = nieuwprijs;  
    this.aankoopprijs = aankoopprijs;  
    this.kmStand = km;  
}
```

Oefening

13 auto's staan stil voor een rood licht.

Wanneer het licht op groen springt, rijden de auto's één voor één weg.

//maak rij voor 13 Auto-objecten

```
Auto[] autos = new Auto[13];
```

//vul rij met auto's

```
for (int i = 0; i < autos.length; i++) {  
    autos[i] = new Auto();  
}
```

Noot:

Klassieke for-lus

//laat elke auto wegrijden

```
for (Auto auto : autos) {  
    auto.rij();  
}
```

Noot:

Alternatieve for-lus

ook datatype 'Verkeerslicht' opnemen

13 auto's staan stil voor een rood licht.

Wanneer het licht op groen springt, rijden de auto's een voor een weg.

```
public class Verkeerslicht {  
    private String kleur;  
  
    public void setKleur(String kleur) {  
        this.kleur = kleur;  
    }  
  
    public String getKleur() {  
        return this.kleur;  
    }  
}
```

```
//initialisatie  
Auto[] autos = new Auto[13];  
Verkeerslicht licht = new Verkeerslicht();  
  
for (int i = 0; i < autos.length; i++) {  
    autos[i] = new Auto();  
}  
  
//eigenlijk testscenario  
if (licht.getKleur().equals("groen")) {  
    for (Auto auto : autos) {  
        auto.rij();  
    }  
}
```

Alternatief

13 auto's staan stil voor een rood licht.

Wanneer het licht op groen springt, rijden de auto's een voor een weg.

```
public class Verkeerslicht {  
    private int kleur;  
    public static final int ROOD = 0;  
    public static final int GROEN = 1;  
    public static final int ORANJE = 2;  
  
    public void setKleur(int kleur) {  
        this.kleur = kleur;  
    }  
  
    public int getKleur() {  
        return this.kleur;  
    }  
}
```

```
//initialisatie  
Auto[] autos = new Auto[13];  
Verkeerslicht licht = new Verkeerslicht();  
  
for (int i = 0; i < autos.length; i++) {  
    autos[i] = new Auto();  
}  
  
//eigenlijk testscenario  
if (licht.getKleur() == Verkeerslicht.GROEN) {  
    for (Auto auto : autos) {  
        auto.rij();  
    }  
}
```

Noot: Nog een ander werkwijze is
via enumeraties (zie verder)

Demo

13 auto's staan stil voor een rood licht.

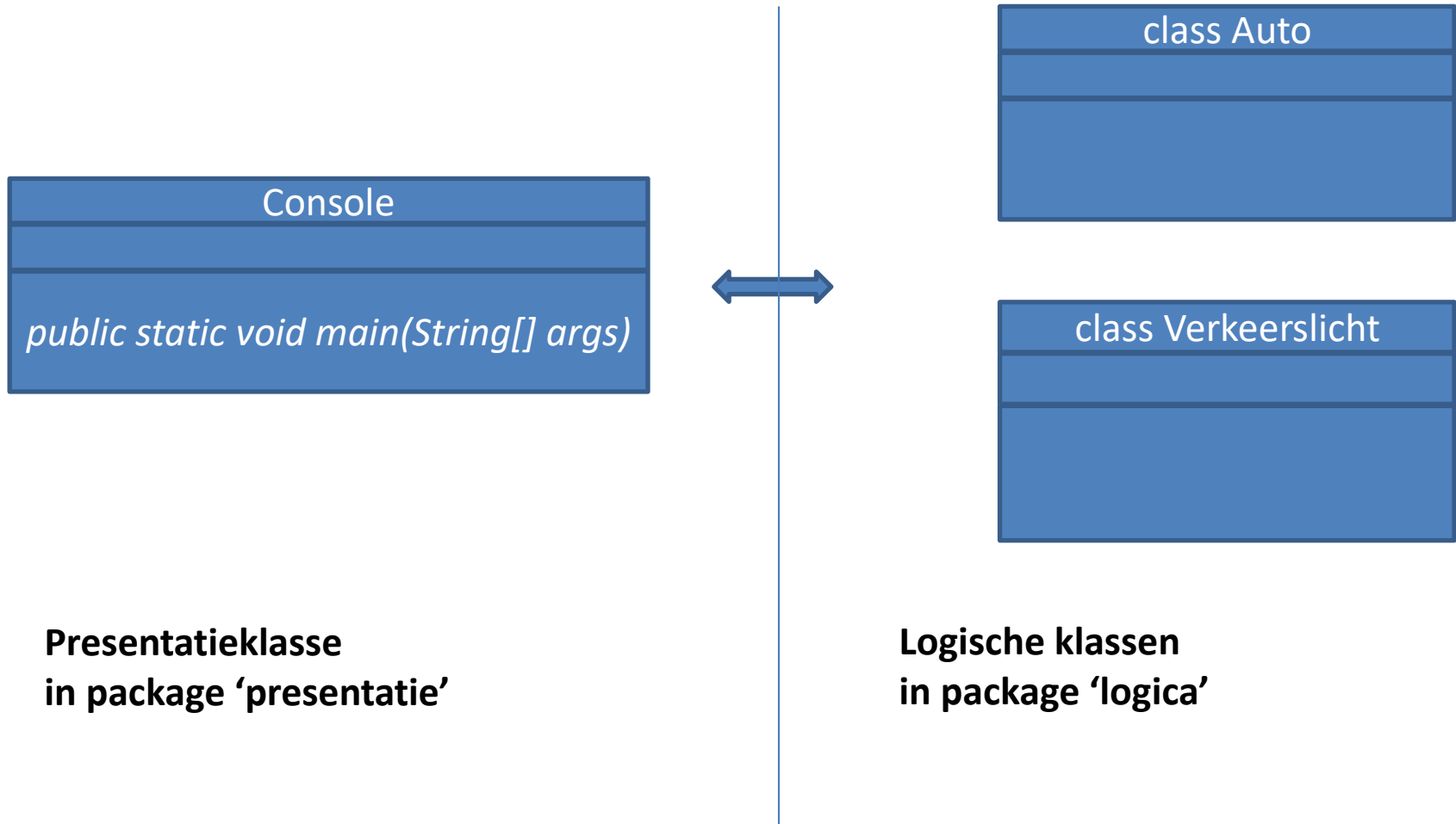
Telkens wanneer het licht op groen springt, kunnen er 5 auto's doorrijden

```
run:
GROEN LICHT
auto_0 rijdt weg
auto_1 rijdt weg
auto_2 rijdt weg
auto_3 rijdt weg
auto_4 rijdt weg
ROOD LICHT
GROEN LICHT
auto_5 rijdt weg
auto_6 rijdt weg
auto_7 rijdt weg
auto_8 rijdt weg
auto_9 rijdt weg
ROOD LICHT
GROEN LICHT
auto_10 rijdt weg
auto_11 rijdt weg
auto_12 rijdt weg
ROOD LICHT
GROEN LICHT
ROOD LICHT
GROEN LICHT
ROOD LICHT
```

```
//eigenlijk testscenario
int cur = 0;
while (true) {
    licht.setKleur(Verkeerslicht.GROEN);
    System.out.println(licht.getLichtstatus());
    for (int i = 0; i < 5; i++) {
        if (cur < autos.length) {
            autos[cur].rij();
            System.out.println("auto_" + (cur++) + " rijdt weg");
        }
    }
    licht.setKleur(Verkeerslicht.ROOD);
    System.out.println(licht.getLichtstatus());

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

Besluit: Samenspel met meerdere logische klassen



packages en import

```
package presentatie;
```

```
import logica.Auto;
```

```
Import logica.Verkeerslicht;
```

```
public class Console {  
    public static void main(String[] args) {  
        //initialisatie  
        Auto[] autos = new Auto[13];  
        Verkeerslicht licht = new Verkeerslicht();  
  
        //eigenlijk testscenario  
        ...  
    }  
}
```

```
package logica;
```

```
public class Auto {  
    ...  
}
```

```
package logica;
```

```
public class Verkeerslicht {  
    ...  
}
```



Enumeraties

Enums in Java API

```
import java.awt.Color;

public class DemoEnumColor {
    public static void main(String[] args) {
        Color c = Color.
    }
}
```

BLACK	Color
BLUE	Color
CYAN	Color
DARK_GRAY	Color
GRAY	Color
GREEN	Color
LIGHT_GRAY	Color
MAGENTA	Color
ORANGE	Color
PINK	Color
RED	Color
WHITE	Color
YELLOW	Color
black	Color
blue	Color
cyan	Color
darkGray	Color

Voorbeeld

```
import java.time.Month;
import java.time.LocalDate;

public class DemoEnumMonth {
    public static void main(String[] args) {
        Month m = Month.FEBRUARY;
        LocalDate today = LocalDate.of(2019, m, 20);
        System.out.println(today);
    }
}
```

```
run:
2019-02-20
BUILD SUCCESSFUL (total time: 2 seconds)
```

```
import java.time.Month;
```

```
public class DemoEnumMonth {
    Month m = Month.FEBRUARY;
}
```

APRIL	Month
AUGUST	Month
DECEMBER	Month
FEBRUARY	Month
JANUARY	Month
JULY	Month
JUNE	Month
MARCH	Month
MAY	Month
NOVEMBER	Month
OCTOBER	Month
SEPTEMBER	Month
from(TemporalAccessor temporal)	Month
of(int month)	Month
valueOf(String arg0)	Month
valueOf(Class<T> enumType, String name) T	
values()	Month[]

Eigen enum definities

Gealloceerd in statische ruimte

```
public enum Geslacht {  
    MAN, VROUW  
}
```

```
public enum Getalstelsel {  
    DECIMAAL, BINAIR, HEXADECIMAAL, OCTAAL  
}
```

```
public enum Weekdag {  
    MAANDAG, DINSDAG, WOENSDAG, DONDERDAG, VRIJDAG  
}
```

```
public enum Kleur {  
    ROOD, GROEN, BLAUW  
}
```

Scope waarbinnen enum geldt

```
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;  
    }  
}
```

```
public class Koe {  
    public enum Kleur {  
        ZWART, BRUIN, PAARS  
    }  
  
    private Kleur kleur;  
  
    public void setKleur(Kleur kleur) {  
        this.kleur = kleur;  
    }  
  
    public Kleur getKleur() {  
        return this.kleur;  
    }  
}
```

```
public enum Kleur {  
    ROOD,  
    ORANJE,  
    GEEL,  
    GROEN,  
    BLAUW,  
    INDIGO,  
    VIOLET  
}
```

13 auto's staan stil voor een rood licht.
Wanneer het licht op groen springt, rijden de auto's een voor een weg.

```
package logica;
```

```
public enum Kleur {  
    ROOD, ORANJE, GEEL,  
    GROEN, BLAUW,  
    INDIGO, VIOLET  
}
```

```
package logica;
```

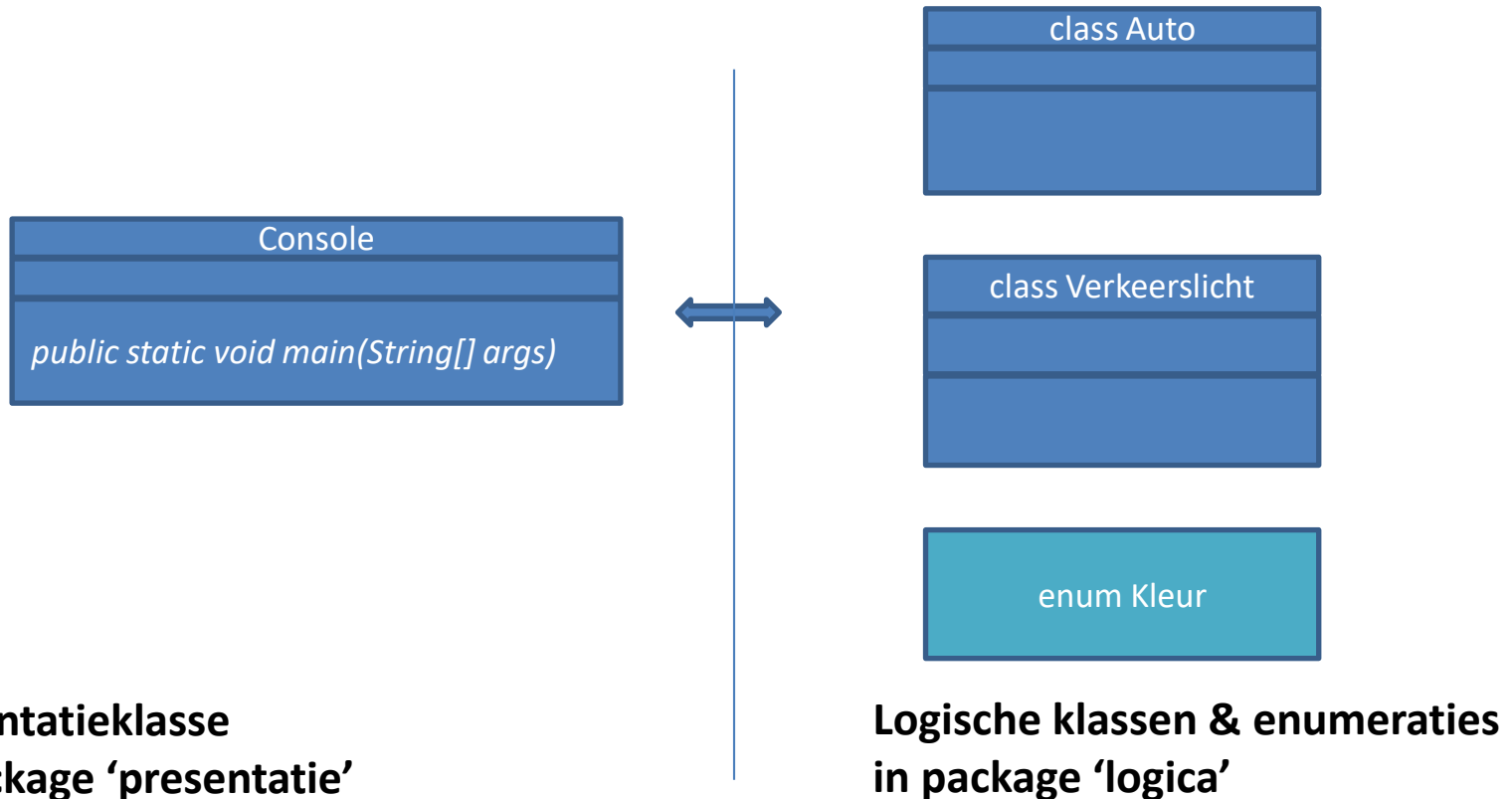
```
public class Verkeerslicht {  
    private Kleur kleur;  
  
    public void setKleur(Kleur kleur) {  
        this.kleur = kleur;  
    }  
  
    public Kleur getKleur() {  
        return this.kleur;  
    }  
}
```

```
package presentatie;
```

```
import logica.Kleur;  
import logica.Verkeerslicht;  
import logica.Auto;
```

```
public class Console {  
    public static void main(String[] args) {  
        Verkeerslicht licht = new Verkeerslicht();  
  
        ...  
        if (licht.getKleur() == Kleur.GROEN) {  
            for (Auto auto : autos) {  
                auto.rij();  
                System.out.println("auto rijdt weg");  
            }  
        }  
    }  
}
```

Besluit: Samenspel met meerdere types



```

package logica;

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;
    }
}

```

```

package presentatie;

import logica.Verkeerslicht;
import logica.Auto;

public class Console {
    public static void main(String[] args) {
        Verkeerslicht licht = new Verkeerslicht();
        ...

        if (licht.getKleur() == Verkeerslicht.Kleur.GROEN) {
            for (Auto auto : autos) {
                auto.rij();
                System.out.println("auto rijdt weg");
            }
        }
    }
}

```

Besluit: Samenspel met meerdere types

