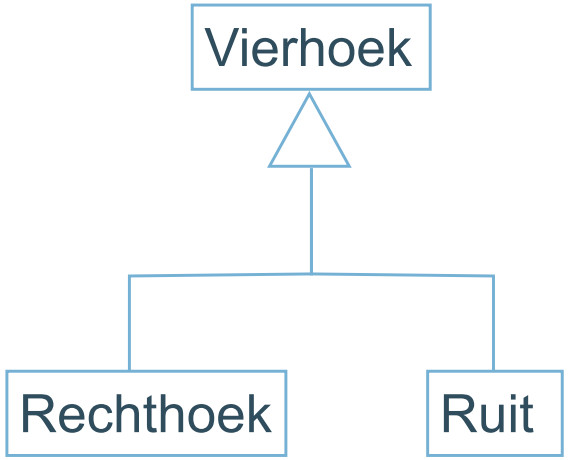
Abstracte klassen & Interfaces

KU LEUVEN

Overerving

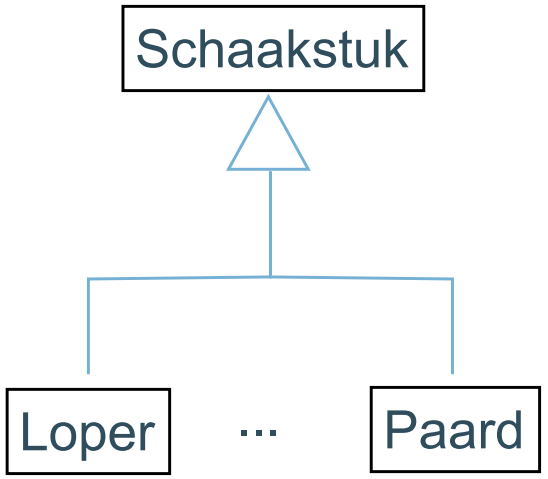
```
public class Vierhoek { ... }

public class Rechthoek extends Vierhoek { ... }

public class Ruit extends Vierhoek { ... }
```



- Elke rechthoek is een vierhoek.
- Elke ruit is een vierhoek.
- Er bestaan ook vierhoeken die geen rechthoeken of ruiten zijn.

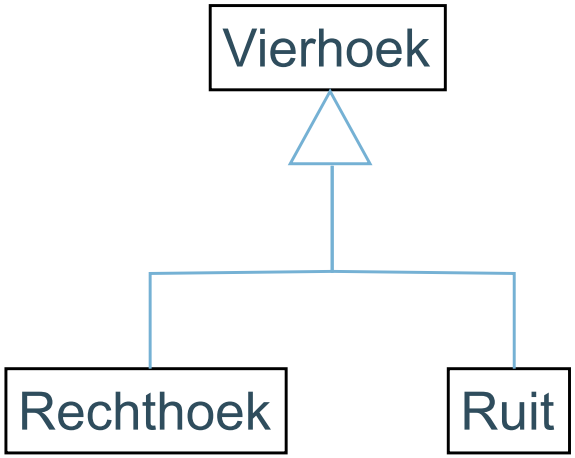
Overerving

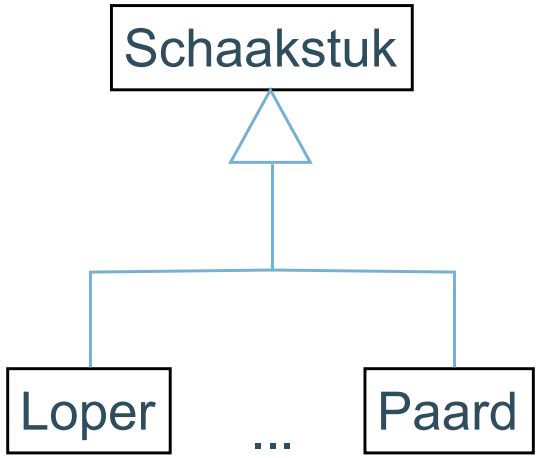
```
public class Schaakstuk { ... }

public class Loper extends Schaakstuk { ... }

public class Paard extends Schaakstuk { ... }
    ...
```



- Elke loper is een schaakstuk
- ...
- Elk paard is een schaakstuk
- Er bestaan geen schaakstukken die geen loper of ... of paard zijn

Verschil



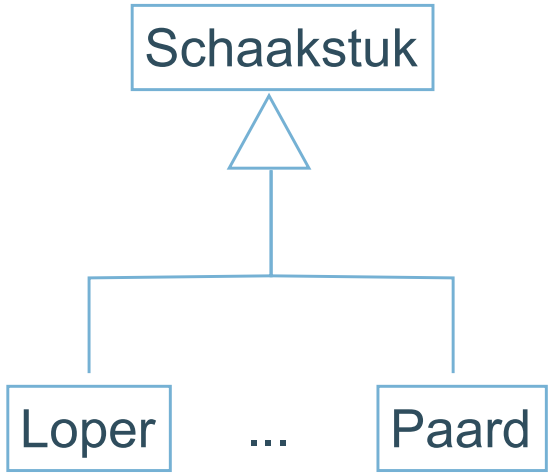


- Vierhoek is concreet concept
- Is groter dan unie van zijn subklassen
- `Vierhoek v = new Vierhoek();`

- Schaakstuk is abstract concept
- Is verzamelnaam voor zijn subklassen
- `Schaakstuk s = new Schaakstuk();`

Abstracte klasse

- Hoewel abstracte klasse eigenlijk zelf “niet bestaat”
- Kan ze toch nuttig zijn in programma
- Gemeenschappelijke functionaliteit van subklassen bevatten
- Gebruiken als type

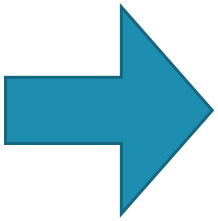


Abstracte klasse

- `public abstract class Schaakstuk { ... }`
- Kan methoden, attributen en constructors bevatten
- Kan gebruikt worden om variabelen te declareren
- Maar je mag er geen objecten van aanmaken
  - Niet: `Schaakstuk s = new Schaakstuk();`
  - Wel: `Schaakstuk s = new Loper();`
- Mag abstracte methoden bevatten

```
public abstract class Schaakstuk {
  private boolean wit;
  public Schaakstuk(boolean w) {
    wit = w;
  }
  public boolean isWit() {
    return wit;
  }
  private SchaakbordVakje positie;
  public SchaakbordVakje getPositie() {
    return positie;
  }
  public void verplaats(SchaakbordVakje nieuw) {
    if (magNaar(nieuw))
        positie = nieuw;
    else System.out.println("Niet doen!");
  }
  public abstract boolean magNaar(SchaakbordVakje nieuw);
}
```





```
public class Loper extends Schaakstuk {
   public Loper(boolean w) {
     super (w);
   }
   public boolean magNaar(SchaakbordVakje nieuw) {
      boolean voorwaarts =
         positie.getRij() - nieuw.getRij()
         == positie.getKolom() - nieuw.getKolom();
      boolean achterwaarts =
         positie.getRij() - nieuw.getRij()
         == nieuw.getKolom() - positie.getKolom();
      boolean verschillend =
        nieuw.getRij() != positie.getRij()
        || nieuw.getKolom() != positie.getKolom();
      return verschillend && (voorwaarts || achterwaarts);
   }
}
```

Niet abstract -> moet alle abstracte methoden implementeren

Abstracte klasse

- Klasse zonder objecten
- Attributen
- Gewone methoden
- Abstracte methoden

KU LEUVEN

Voorbeeld

Vind de jongste werknemer:

```
public static Werknemer jongste(Werknemer[] rij) {
    Werknemer jongste = rij[0];
    for (int i = 1; i < rij.length; i++)
        if (rij[i].getLeeftijd() < jongste.getLeeftijd())
            jongste = rij[i];
    return jongste;
}
```

Vind de kleinste rechthoek:

```
public static Rechthoek kleinste(Rechthoek[] rij) {
    Rechthoek kleinste = rij[0];
    for (int i = 1; i < rij.length; i++)
        if (rij[i].getOpp() < kleinste.getOpp())
            kleinste = rij[i];
    return kleinste;
}
```

Met abstracte klassen

```
public abstract class Vergelijkbaar {

    public abstract boolean kleinerDan(Vergelijkbaar v);

     public static Vergelijkbaar kleinste(Vergelijkbaar[] rij) {
       Vergelijkbaar kleinste = rij[0];
       for (int i = 1; i < rij.length; i++)
           if (rij[i].kleinerDan(kleinste))
               kleinste = rij[i];
       return kleinste;
     }

}
```

Met abstracte klassen

```
public abstract class Vergelijkbaar {
    public abstract boolean kleinerDan(Vergelijkbaar v);
     public static Vergelijkbaar kleinste(Vergelijkbaar[] rij) {
       Vergelijkbaar kleinste = rij[0];
       for (int i = 1; i < rij.length; i++)
           if (rij[i].kleinerDan(kleinste))
               kleinste = rij[i];
       return rij[i];
     }
}

public class Werknemer extends Vergelijkbaar { …
    public boolean kleinerDan(Vergelijkbaar v) {
        if (!v instanceof Werknemer) { ... }
        return this.getLeeftijd() < ((Werknemer) v).getLeeftijd();
    }
    …
}
```

Met abstracte klassen

```
public abstract class Vergelijkbaar {
    public abstract boolean kleinerDan(Vergelijkbaar v);
    public static Vergelijkbaar kleinste(Vergelijkbaar[] rij) {
        Vergelijkbaar kleinste = rij[0];
        for (int i = 1; i < rij.length; i++)
            if (rij[i].kleinerDan(kleinste))
                kleinste = rij[i];
        return rij[i];
    }
}

public class Rechthoek extends Vergelijkbaar { …
    public boolean kleinerDan(Vergelijkbaar v) {
        if (!v instanceof Rechthoek) { ... }
        return this.oppervlakte() < ((Rechthoek) v).oppervlakte();
    }
  …
}
```

Probleem

Een klasse kan maar één superklasse hebben

~~public class Rechthoek extends Vergelijkbaar, Vierhoek { ... }~~

Oplossing: **interface**

```
public interface Vergelijkbaar {

    public boolean kleinerDan(Vergelijkbaar v);

}

public class Rechthoek extends Vierhoek implements Vergelijkbaar { ... }
```

Abstracte klasse

- Klasse zonder objecten
- Gewone methoden, attributen
- Abstracte methoden
- Slechts overerven van één

Interface

- “Klasse” zonder objecten
- “Abstracte” methoden
- Willekeurig aantal implementeren

Richtlijn: is het essentiële eigenschap of niet?

Analyse & Ontwerp

KU LEUVEN

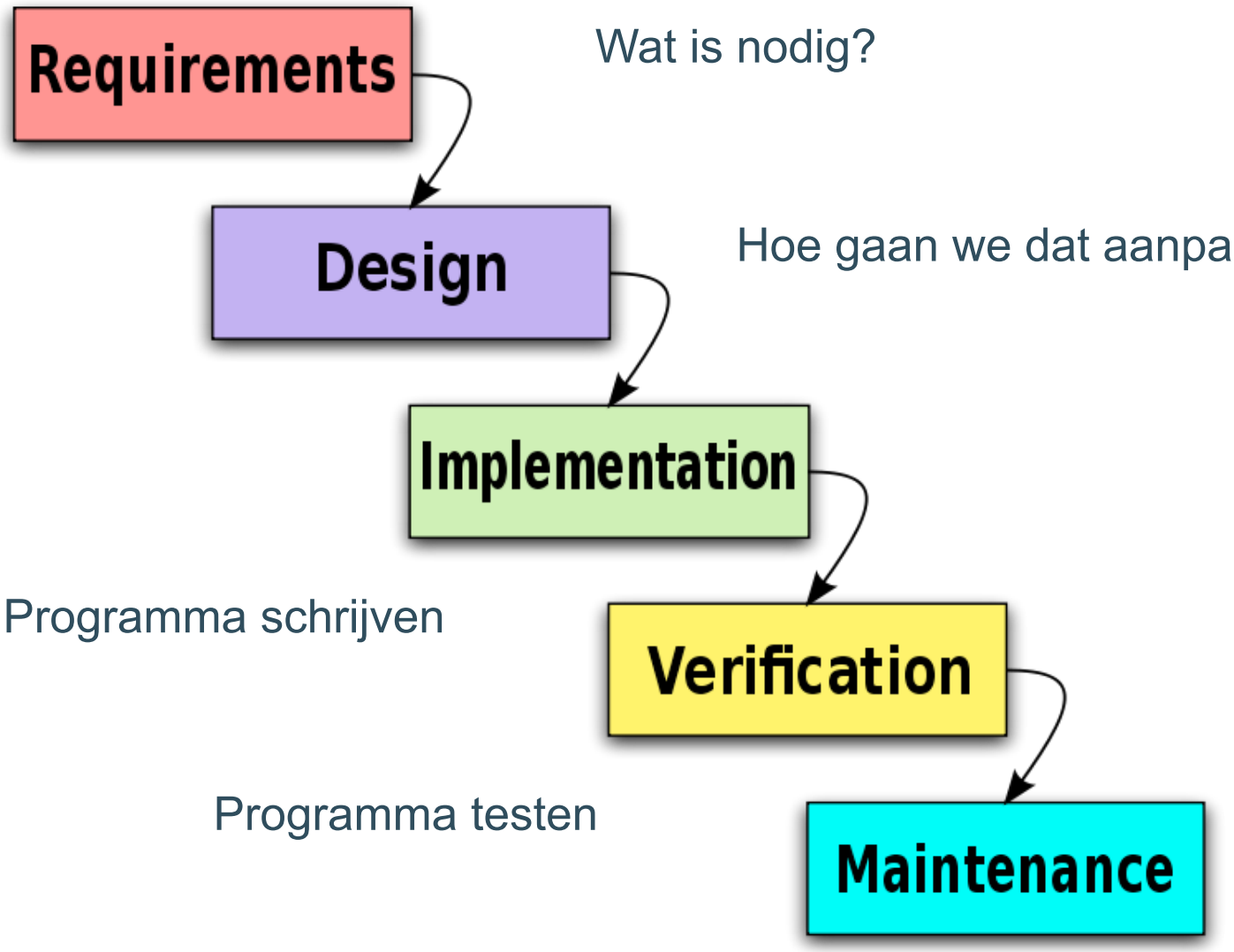
Probleem



???



Waterval (1970)



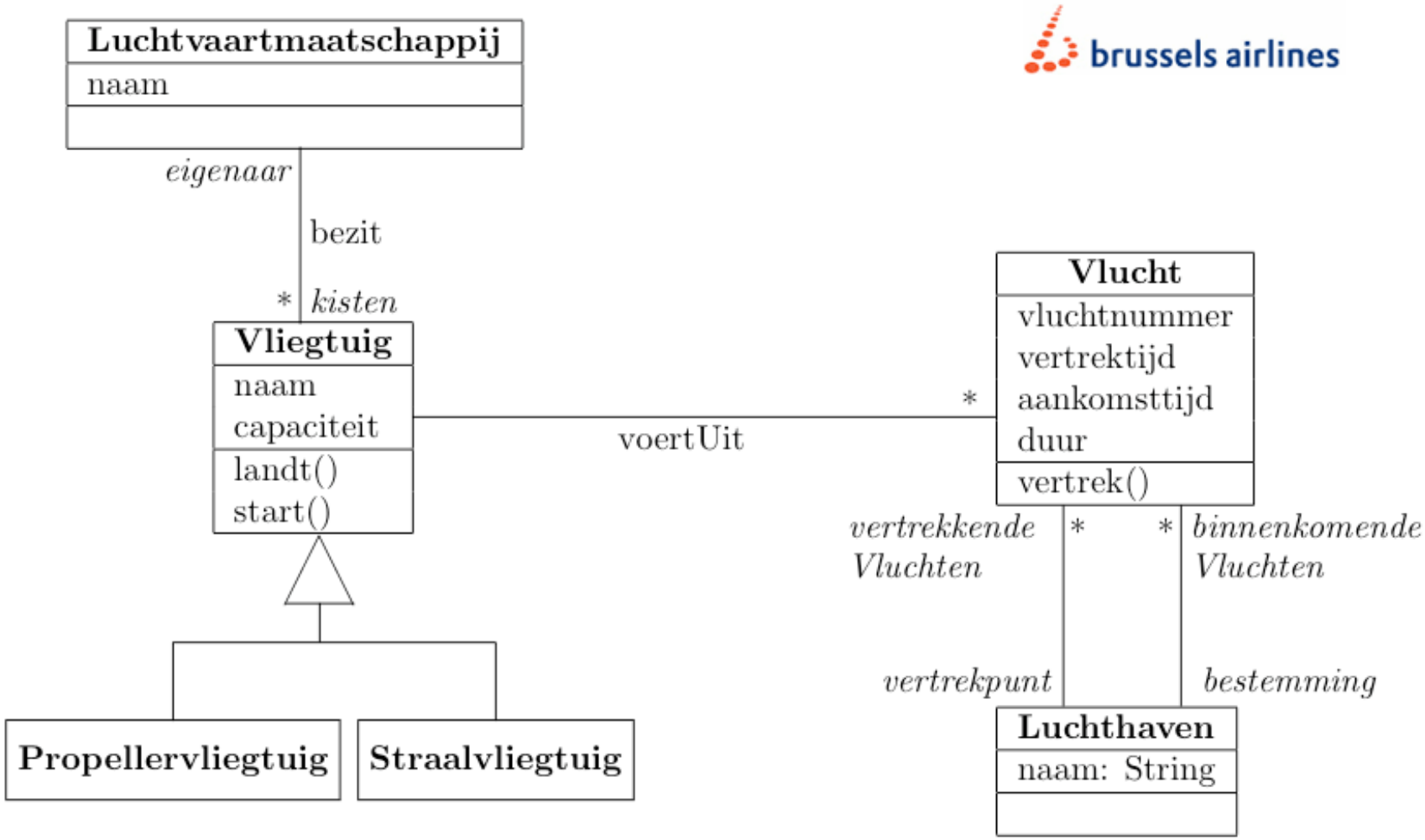
Agile


AGILE
1
Requirements
2
Design
3
Development
4
Testing
5
Deployment
6
Review

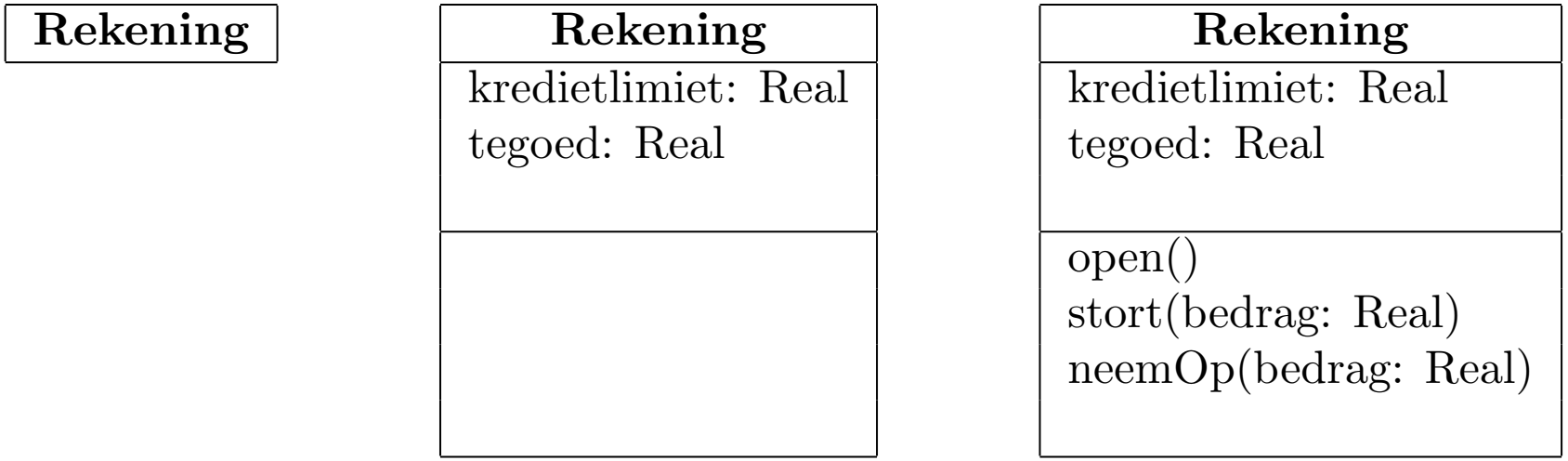
UML: Hulpmiddelen voor analyse & ontwerp



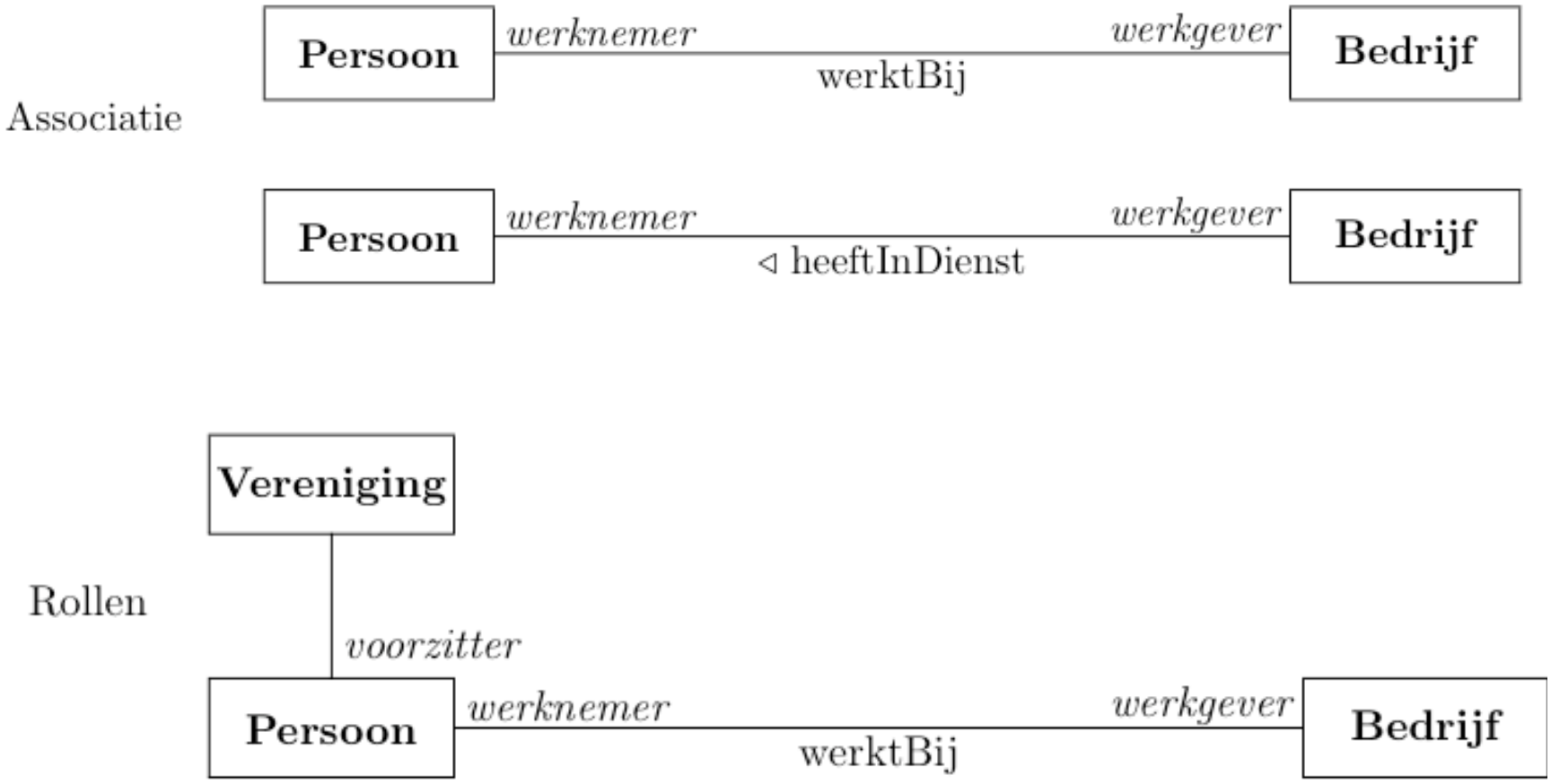
Klassediagramma



Details: Klassen

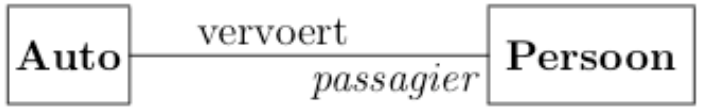


Details: Associaties en rollen

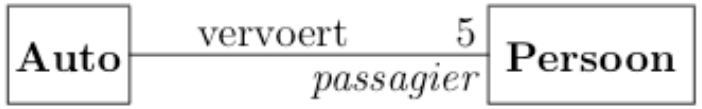


Details: multipliciteit

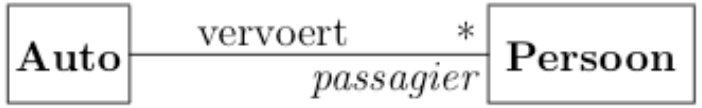
Multipliciteit



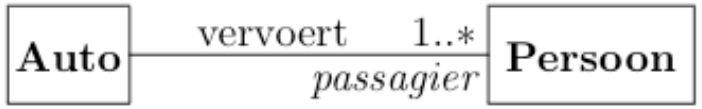
Auto vervoert exact één *Persoon*



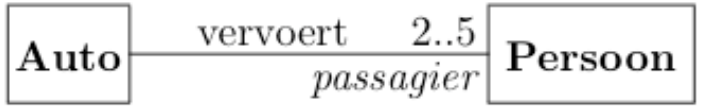
Auto vervoert exact vijf *Personen*



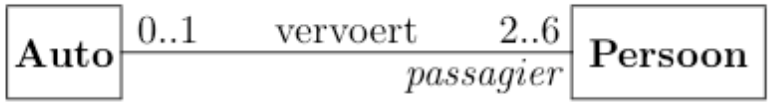
Auto vervoert nul of meer *Personen*



Auto vervoert één of meer *Personen*



Auto vervoert twee tot vijf *Personen*

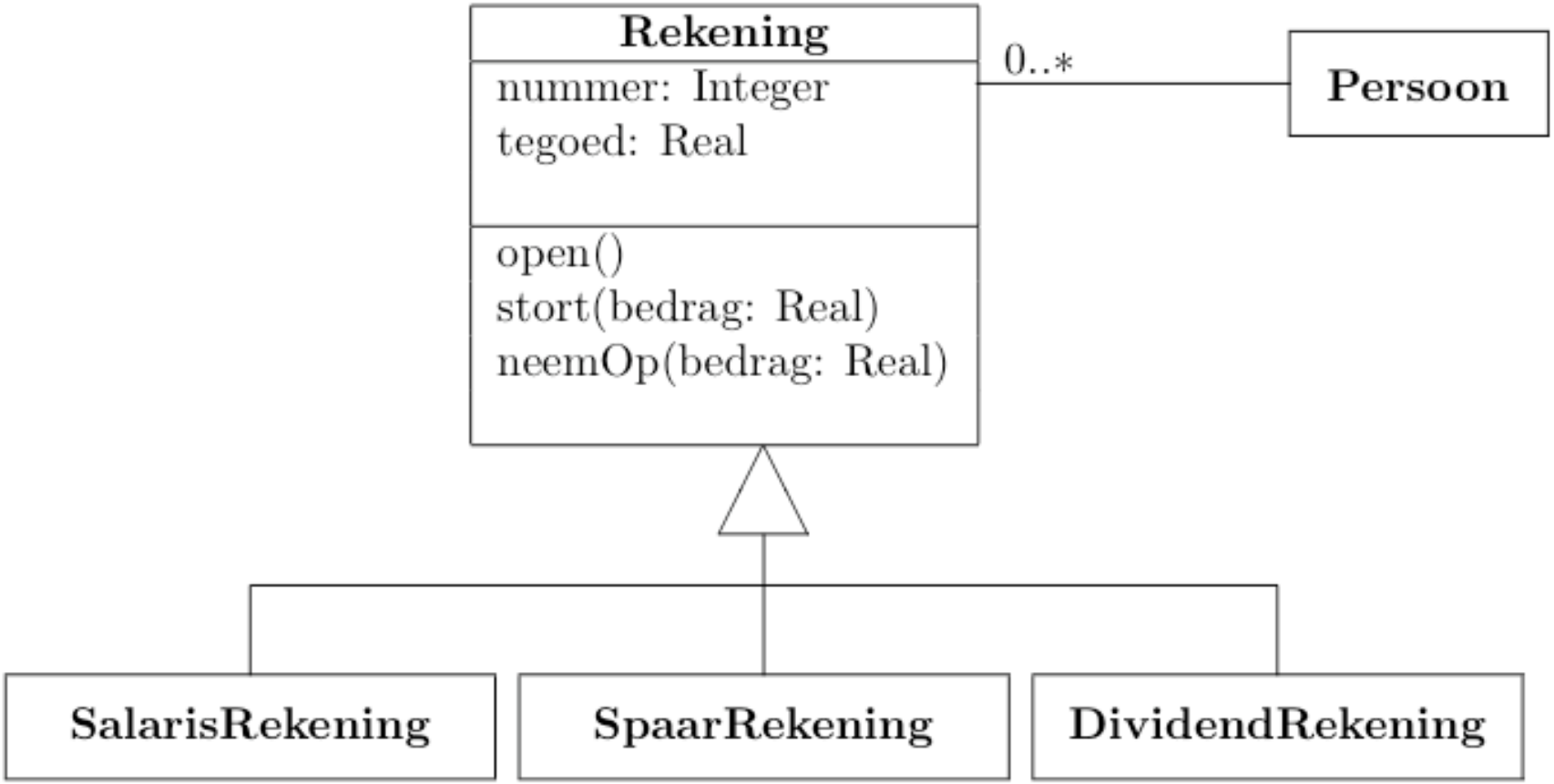


Auto vervoert twee tot zes *Personen*

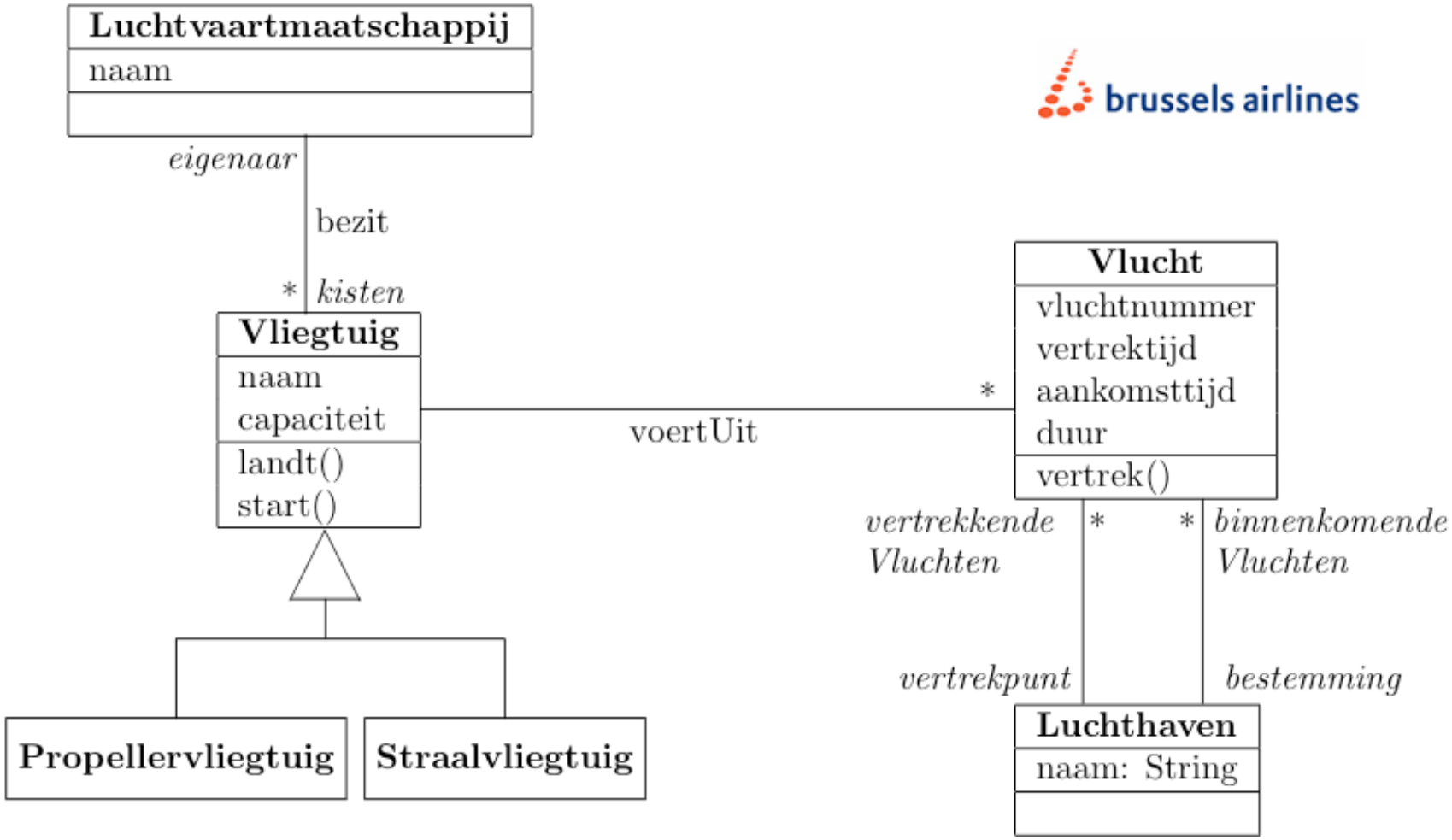
Persoon wordt vervoerd door nul of één *Auto*

KU LEUVEN

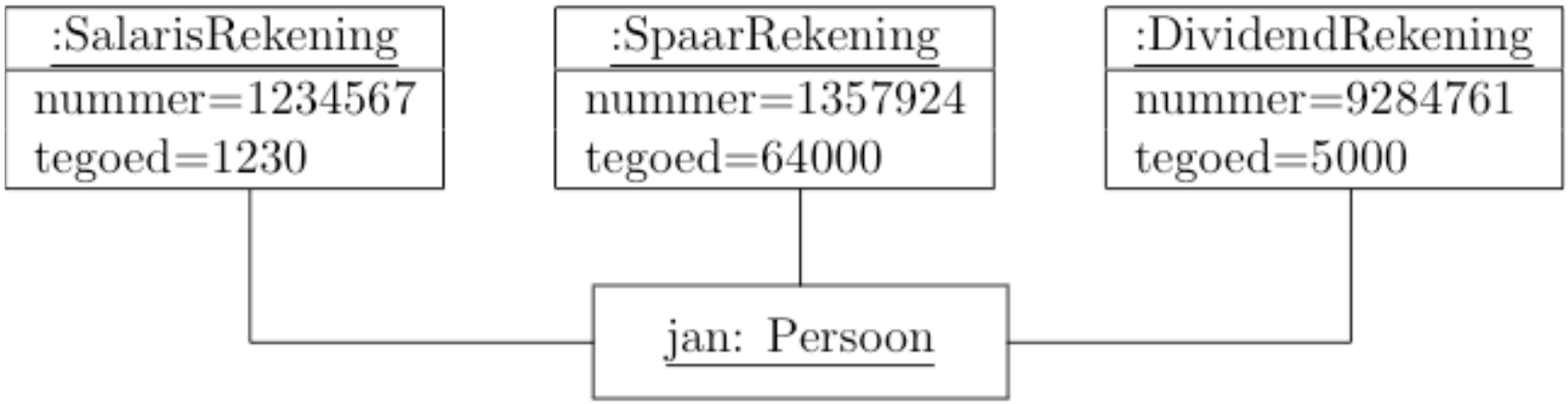
Details: subklassen



Klassediagramma



Objectdiagramma



Concrete instantiatie van klassediagramma op bepaald tijdstip

Zowel bij analyse als ontwerp

- Klassen/objecten zoals ze voorkomen in het domein
- Klassen/objecten zoals ze voorkomen in je ontwerp

Pakketten

KU LEUVEN

Pakketten

- Verzameling van klassen die samen een functionaliteit aanbieden
- Toegangsniveau
- Filosofie van Java:
  - Zoveel mogelijk details afschermen

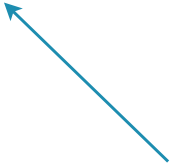


Gebruik

```
package pakket1;

public class Klasse {
...
}
```

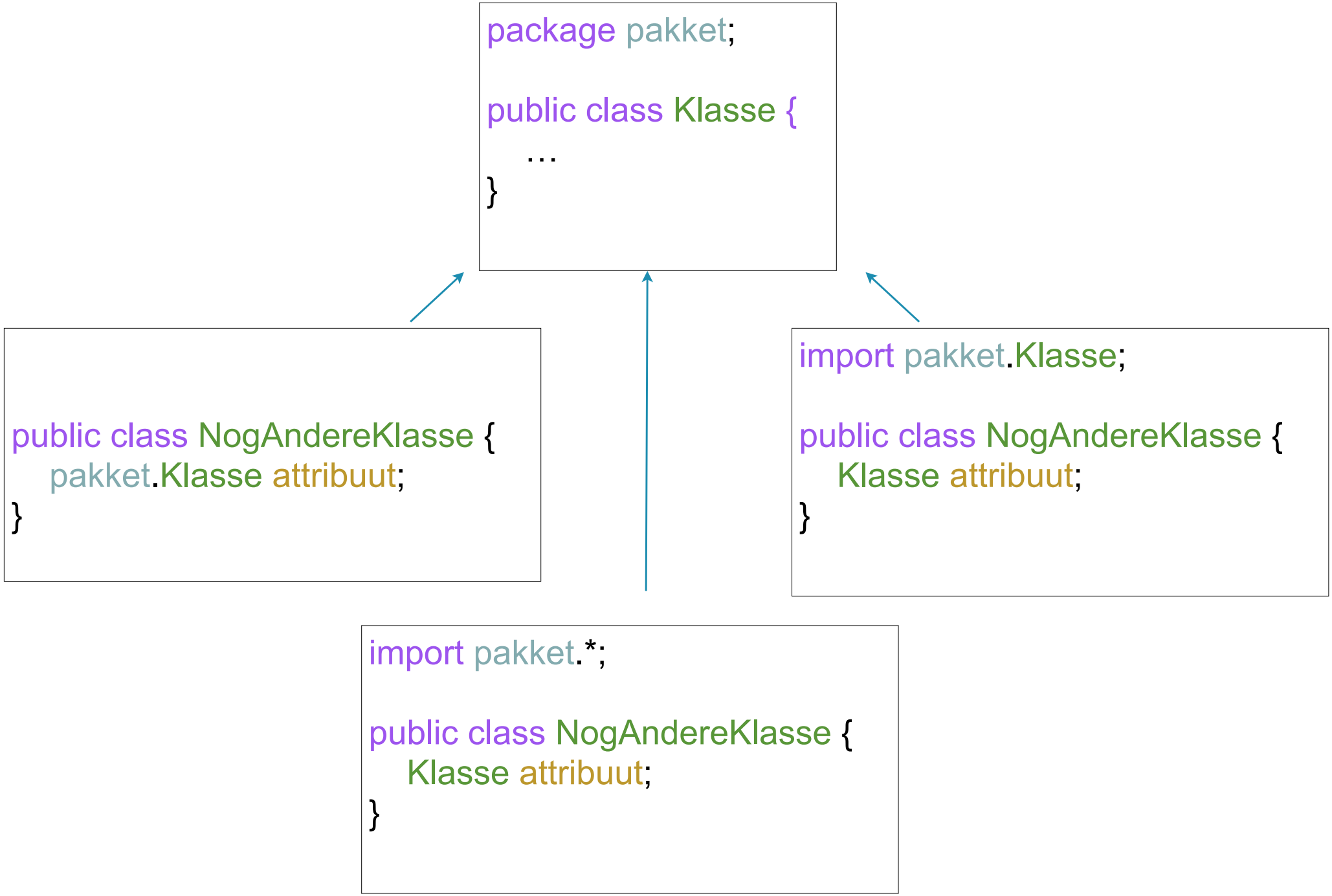
```
package pakket2;

public class NogAndereKlasse {
    pakket1.Klasse attribuut;
}
```





```
package pakket1;

public class AndereKlasse {
    Klasse attribuut;
}
```

```
public class NogAndereKlasse {
    pakket1.Klasse attribuut;
}
```


package pakket;
public class Klasse {
…
}
public class NogAndereKlasse {
pakket.Klasse attribuut;
}
import pakket.Klasse;
public class NogAndereKlasse {
Klasse attribuut;
}
import pakket.*;
public class NogAndereKlasse {
Klasse attribuut;
}

Bestanden

- Pakket = directory
- Klasse = bestand
- import foo.bar.Bla;
- BASE/foo/bar/Bla.class



Pakkettoegankelijk

- Methoden/attributen/klassen
- kunnen enkel binnen hetzelfde pakket toegankelijk zijn
- aangegeven door **afwezigheid** van **public**, **private** of **protected**



Pakkettoegankelijk

```
package pakket;

public class Klasse {
    int getal;
}
```




```
package pakket;

public class AndereKlasse {
    public static void main(String[] args) {
        (new Klasse()).getal = 5;
    }

}
```

```
public class NogAndereKlasse {
    public static void main(String[] args) {
        (new Klasse()).getal = 5;
    }
}
```

Ondersteuning voor encapsulatie



- Toegansniveaus
  - private: voor niemand
  - niets: voor klassen uit zelfde pakket
  - protected: voor subklassen of uit hetzelfde pakket
  - public: voor iedereen

Standaard pakketten

- java.lang
  - Object, System, Integer, ...
- java.io
- java.awt
- javax.swing
- java.net

Inlezen

```
import java.io.*;

class PersonalHello {
    public static void main (String args[])   {
        byte[] name = new byte[100];
        int nr_read = 0;
        System.out.println("What is your name?");
        try {
            nr_read = System.in.read(name);
            System.out.print("Hello ");
            System.out.write(name,0,nr_read);
        }
        catch (IOException e) {
            System.out.print("I'm Sorry.  I didn't catch your name.");
        }
    }
}
```

Inlezen (2)

```
import java.io.*;

class Addition {
    public static void main (String[] args) throws IOException {
        BufferedReader stdin = new BufferedReader(
                        new InputStreamReader(System.in) );
        int num1, num2;
        System.out.print ("Enter an integer number: ");
        num1 = Integer.parseInt(stdin.readLine());
        System.out.print ("Enter another integer number: ");
        num2 = Integer.parseInt(stdin.readLine());
        System.out.println ("The sum is " + (num1+num2));
    }
}
```