14. Java Klassen

Klassen, Typen, Objekte, Deklaration, Instanzierung, Konstruktoren, statische Felder und Methoden , Datenkapselung

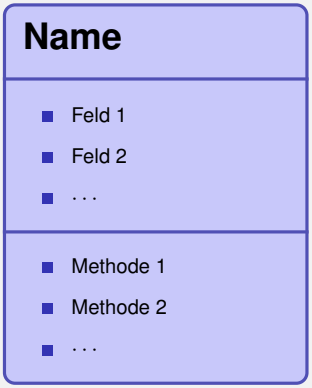
Klassen (Java) vs. Records (Pascal)

Pascal	Java
RECORDs in Pascal sind reine *Datenobjekte*. Auf ihnen wird mit Prozeduren operiert.	Klassen in Java beherbergen *Daten und Code*. Sie stellen Code bereit, mit dem auf ihnen operiert wird.
RECORDs sind *Werte*: Instanzen werden automatisch alloziert.	Klassen sind *Referenzen*: Instanzen müssen mit `new` alloziert werden. Instanzen heissen Objekte.

Klassen - Technisch

Eine Klasse ist eine Einheit mit einem *Namen*, die *Daten* und *Funktionalität* beinhaltet

- Die Klasse definiert einen neuen *Datentyp*.
- *Daten* sind Variablen und heissen *Felder* oder *Attribute*.
- *Funktionalität* ist vorhanden in Form von *Methoden*, die in der Klasse definiert sind.
- Klassen sind separate `.java` Dateien mit gleichem Namen



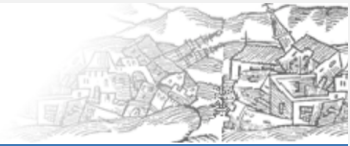
Klassen - Konzeptuell

Klassen erlauben es, Daten, die inhaltlich *zusammengehören*, zu einem Datentyp *zusammenzufassen*.

Klassen bieten Funktionalitäten an, welche *Abfragen* basierend auf den Daten oder *Operationen* auf den Daten ermöglichen.

Klassen - Beispiel: Erdbebendaten

Schweizerischer Erdbebendienst
Service Sismologique Suisse
Servizio Sismico Svizzero
Swiss Seismological Service



SED > *Earthquake catalog* > Query the catalogue

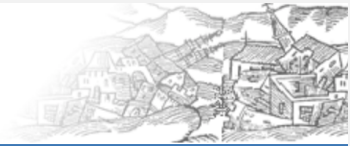
Earthquake catalog

link	date	time	appraisal	event type	lat [°N]	lon [°E]	source agencygency	depth	Mw	Ml	Io	Ix	epicentral area
»	2001/01/01	00:03:47.8	certain	earthquake	45.53	6.75	RENASS/BCSF (2009)	5.	1.52	0.9			SSE BEAUFORT(73)
»	2001/01/01	00:20:01.5	uncertain	earthquake	47.51	9.48	LED (2009)	10.	2.17	1.99			
»	2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36	2.3			
»	2001/01/07	18:55:18.3	certain	earthquake	48.05	9.03	LED (2009)	15.	1.82	1.41			
»	2001/01/07	20:55:36.5	certain	earthquake	46.564	10.29	SED (ECOS-	5.	1.94	1.6			

http://hitseddb.ethz.ch:8080/ecos09/result.html?tremors=earthquake&time_start=2001&time_end=2008

Klassen - Beispiel: Erdbebendaten

Schweizerischer Erdbebendienst
Service Sismologique Suisse
Servizio Sismico Svizzero
Swiss Seismological Service



SED > *Earthquake catalog* > Query the catalogue

Earthquake catalog

link	date	time	appraisal	event type	lat [°N]	lon [°E]	source agencygency	depth	Mw	Ml	Io	Ix	epicentral area
»	2001/01/01	00:03:47.8	certain	earthquake	45.53	6.75	RENASS/BCSF (2009)	5.	1.52	0.9			SSE BEAUFORT(73)
»	2001/01/01	00:20:01.5	uncertain	earthquake	47.51	9.48	LED (2009)	10.	2.17	1.99			
»	2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36	2.3			
»	2001/01/07	18:55:18.3	certain	earthquake	48.05	9.03	LED (2009)	15.	1.82	1.41			
»	2001/01/07	20:55:36.5	certain	earthquake	46.564	10.29	SED (ECOS-	5.	1.94	1.6			

http://hitseddb.ethz.ch:8080/ecos09/result.html?tremors=earthquake&time_start=2001&time_end=2008

Klasse für Messwert - Erster Versuch

date	time	appraisal	event type	lat [°N]	lon [°E]	source agencygency	depth	Mw
2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36

Klasse für Messwert - Erster Versuch

date	time	appraisal	event type	lat [°N]	lon [°E]	source agencygency	depth	Mw
2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36

Datei `Messwert.java`:

```
public class Messwert {

    String datum;
    String zeit;

    float breitengrad;
    float laengengrad;

    float magnitude;
}
```

Klasse für Messwert - Erster Versuch

date	time	appraisal	event type	lat [°N]	lon [°E]	source agencygency	depth	Mw
2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36

Datei `Messwert.java`:

```
public class Messwert {

    String datum;
    String zeit;

    float breitengrad;
    float laengengrad;

    float magnitude;
}
```

Name der Klasse/ des Datentyps

Klasse für Messwert - Erster Versuch

date	time	appraisal	event type	lat [°N]	lon [°E]	source agencygency	depth	Mw
2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36

Datei `Messwert.java`:

```
public class Messwert {

    String datum;
    String zeit;

    float breitengrad;
    float laengengrad;

    float magnitude;
}
```

Felder gemäss Tabellenkopfdaten

Klasse für Messwert - Erster Versuch

date	time	appraisal	event type	lat [°N]	lon [°E]	source agencygency	depth	Mw
2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36

Datei `Messwert.java`:

```
public class Messwert {

    String datum;
    String zeit;

    float breitengrad;
    float laengengrad;

    float magnitude;
}
```

Objekte: Instanzen von Klassen

Klassen beschreiben den Aufbau von Objekten, eine Art *Bauplan*

$\Rightarrow$ Vergleichbar mit den *Kopfdaten* aus der Tabelle.

Objekte werden instanziert nach Bauplan und enthalten nun Werte.

$\Rightarrow$ Vergleichbar mit den einzelnen Datenzeilen aus der Tabelle.

Objektinstanzierung: Das Schlüsselwort `new`

```
Messwert w;
```

Objektinstanzierung: Das Schlüsselwort `new`

Variable “w” vom Typ “Messwert”

```
Messwert w;
```

Objektinstanzierung: Das Schlüsselwort `new`

Variable “w” vom Typ “Messwert”

```
Messwert w;
```

Objektinstanzierung: Das Schlüsselwort `new`

```
Messwert w;

w = new Messwert();
```

Objektinstanzierung: Das Schlüsselwort `new`

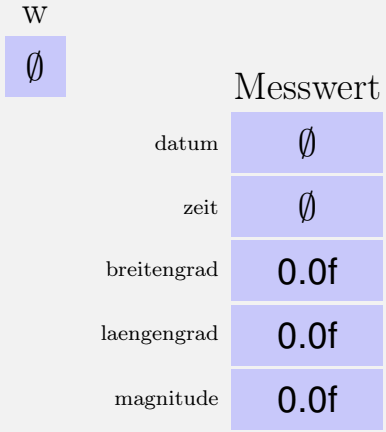
```
Messwert w;

w = new Messwert();
```

Instanzierung eines Objekts vom Typ Messwert

Objektinstanzierung: Das Schlüsselwort `new`

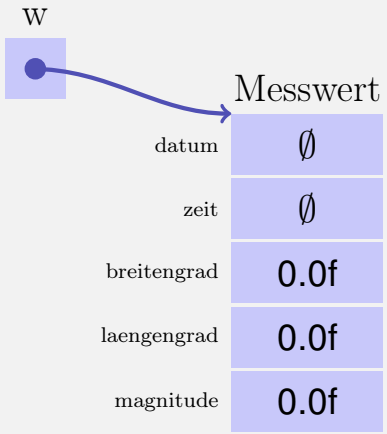
```
Messwert w;
w = new Messwert();
```

Instanzierung eines Objekts vom Typ Messwert



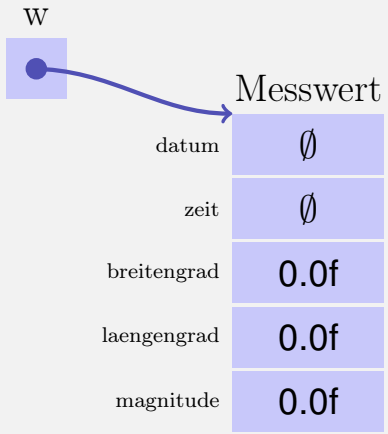
Objektinstanzierung: Das Schlüsselwort `new`

```
Messwert w;

w = new Messwert();
```



Dereferenzierung: Zugriff auf Felder

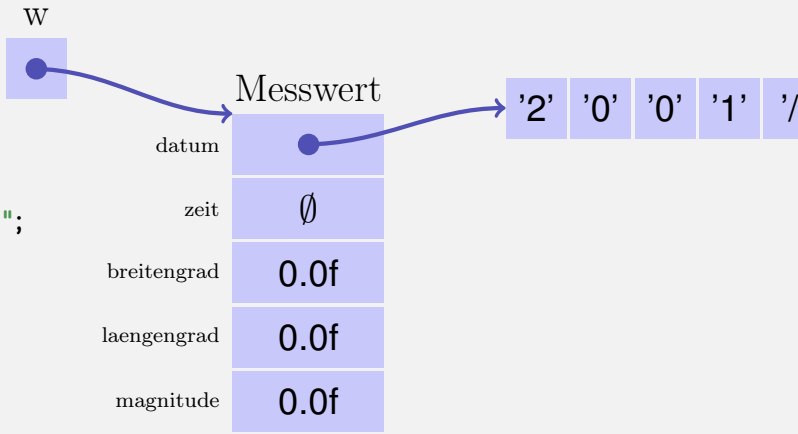
```
Messwert w;

w = new Messwert();
```



Dereferenzierung: Zugriff auf Felder

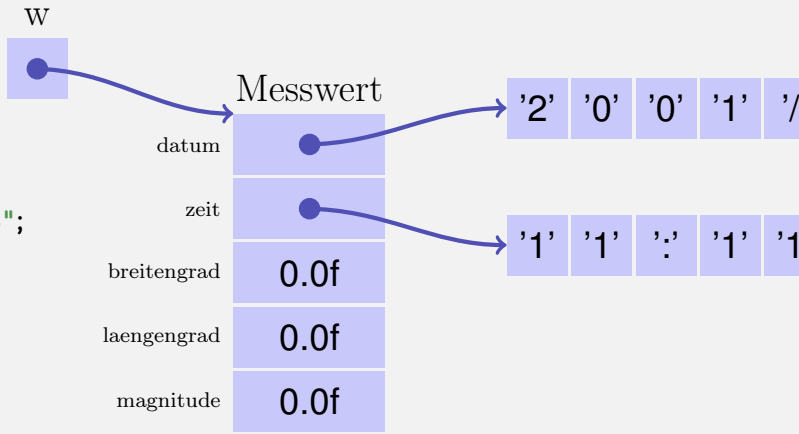
```
Messwert w;

w = new Messwert();

w.datum = "2001/01/03";
```

Dereferenzierung: “Dem Pfeil von w folgen”



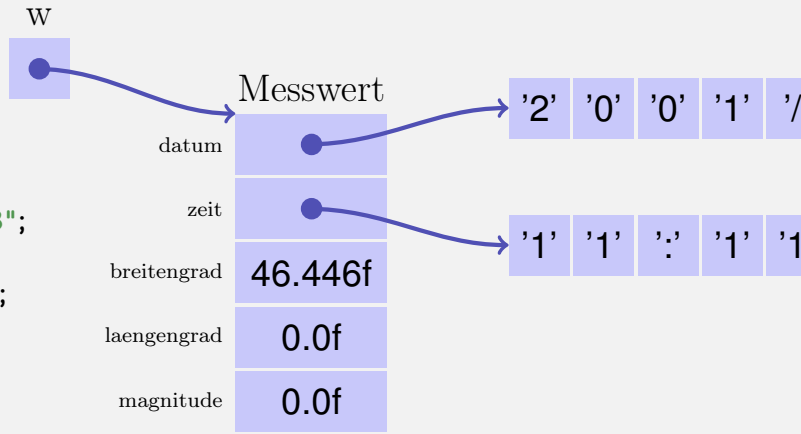
Dereferenzierung: Zugriff auf Felder

```
Messwert w;

w = new Messwert();

w.datum = "2001/01/03";
w.zeit  = "11:11:20.4";
```



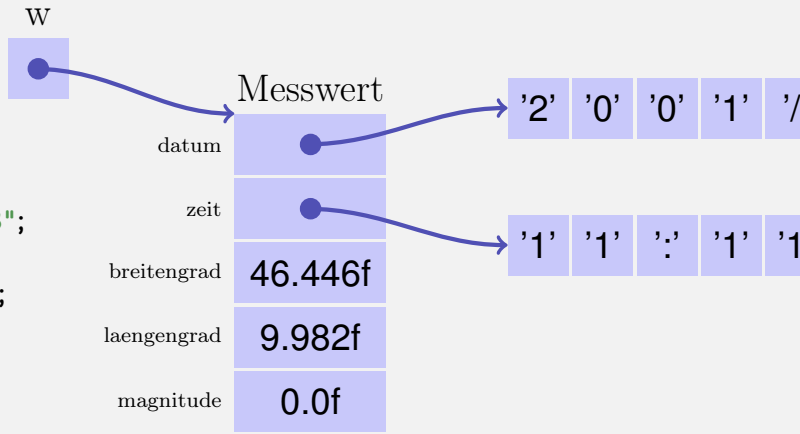
Dereferenzierung: Zugriff auf Felder

```
Messwert w;

w = new Messwert();

w.datum = "2001/01/03";
w.zeit = "11:11:20.4";
w.breitengrad = 46.446f;
```



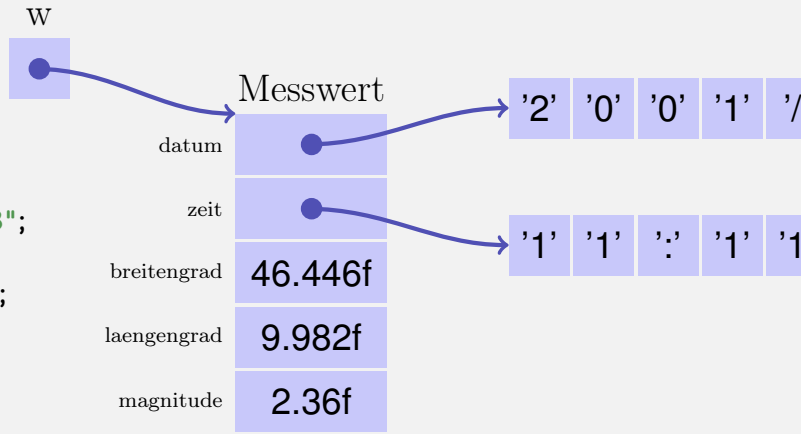
Dereferenzierung: Zugriff auf Felder

```
Messwert w;

w = new Messwert();

w.datum = "2001/01/03";
w.zeit = "11:11:20.4";
w.breitengrad = 46.446f;
w.laengengrad = 9.982f;
```



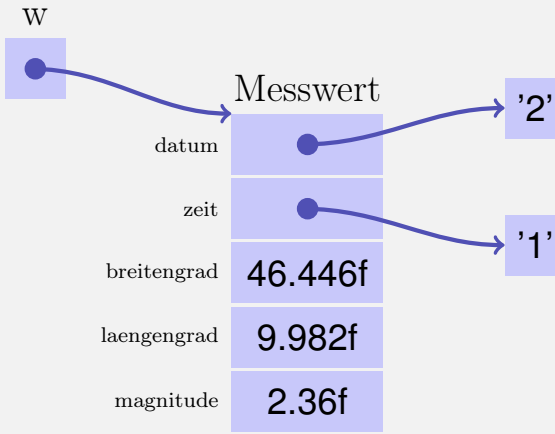
Dereferenzierung: Zugriff auf Felder

```
Messwert w;

w = new Messwert();

w.datum = "2001/01/03";
w.zeit = "11:11:20.4";
w.breitengrad = 46.446f;
w.laengengrad = 9.982f;
w.magnitude = 2.36f;
```



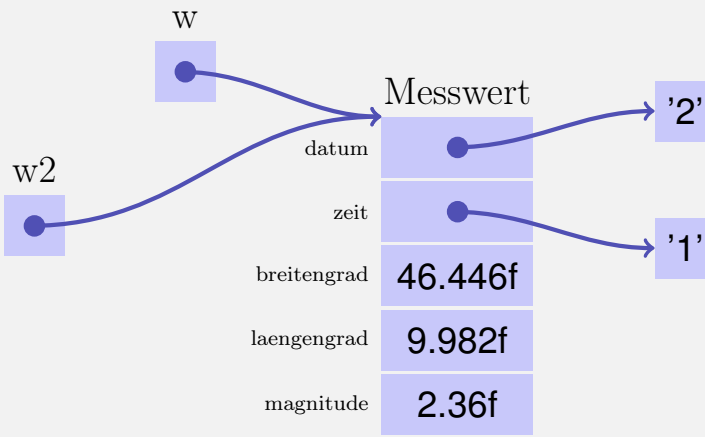
Objekte sind Referenztypen: Aliasing

```
Messwert w;

w = new Messwert();
```



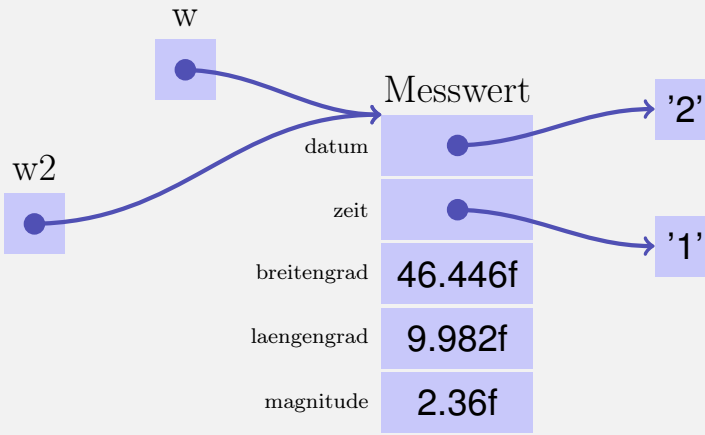
Objekte sind Referenztypen: Aliasing

```
Messwert w;

w = new Messwert();

Messwert w2 = w;
```



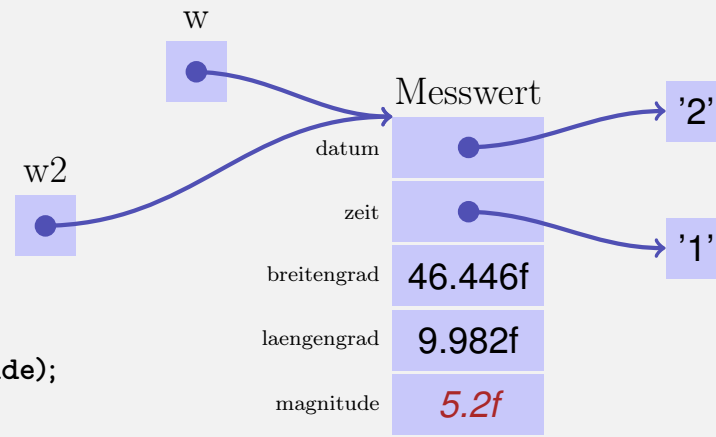
Objekte sind Referenztypen: Aliasing

```
Messwert w;

w = new Messwert();

Messwert w2 = w;

w2.magintude = 5.2f;
```



Objekte sind Referenztypen: Aliasing

```
Messwert w;

w = new Messwert();

Messwert w2 = w;

w2.magintude = 5.2f;

System.out.println(w.magnitude);
```



Moment mal! ...

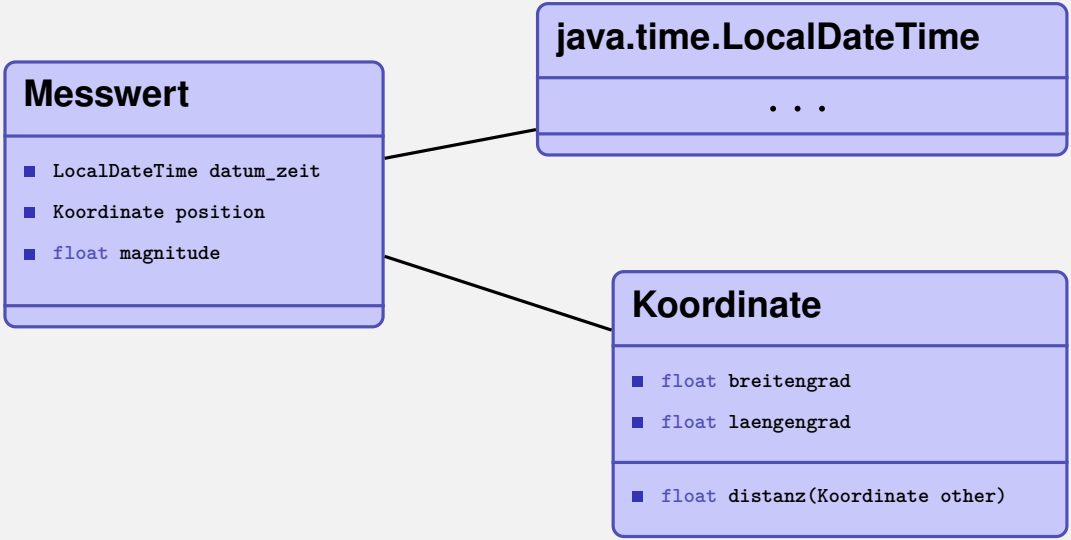
Klassen erlauben es, Daten, die inhaltlich *zusammengehören*, zu einem Datentyp *zusammenzufassen*.

Ist das ein gutes Klassendesign?

Es geht vielleicht besser!

- Datum und Zeit gehören gehören in eine eigene Klasse: Java bietet das schon an: `java.time.LocalDateTime`
- Breitengrad und Längengrad gehören in einen Datentyp `Koordinate`.

Klasse für Messwert: Zweiter Versuch



Methoden in Klassen

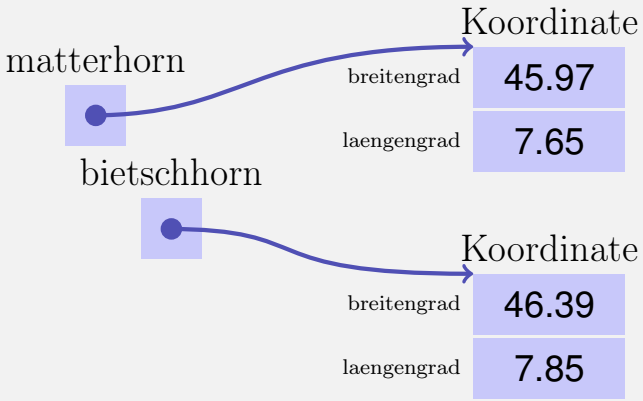
```
public class Koordinate {

    float breitengrad;
    float laengengrad;

    /**
     * Berechnet die Distanz zur uebergebenen
     * Koordinate 'other' in Kilometer.
     */
    float distanz(Koordinate other){
        float dl = this.laengengrad - other.laengengrad;
        // Rest: Entfernungsberechnung der Orthodrome
    }
}
```

Methoden in Klassen

```
public class Koordinate {

    float breitengrad;
    float laengengrad;

    /**
     * Berechnet die Distanz zur uebergebenen
     * Koordinate 'other' in Kilometer.
     */
    float distanz(Koordinate other){
        float dl = this.laengengrad - other.laengengrad;
        // Rest: Entfernungsberechnung der Orthodrome
    }
}
```

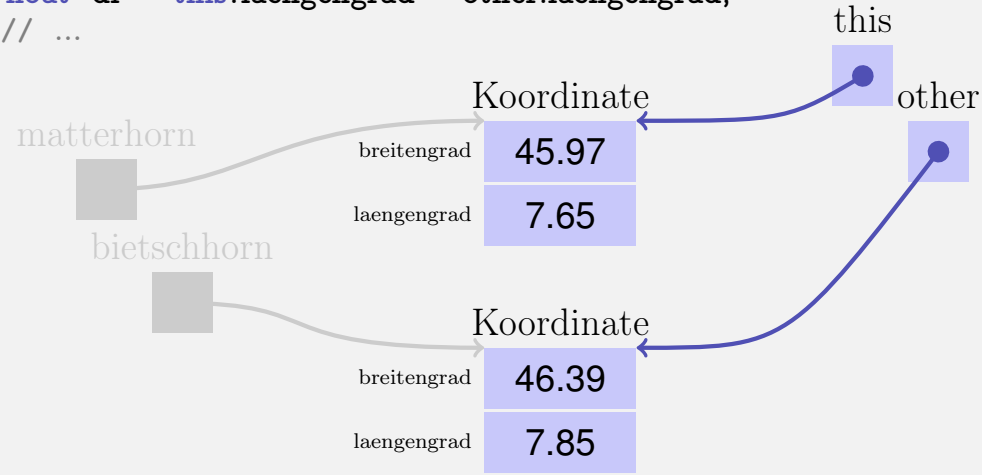
Methodenaufruf - Setting

```
Koordinate matterhorn, bietschhorn;
// ... Instanzieren und Setzen von Werten
d = matterhorn.distanz(bietschhorn);
```



Im Kontext der Methode

```
float distanz(Koordinate other){
        float dl = this.laengengrad − other.laengengrad;
        // ...
```



Schlüsselwort `this`

`this` erlaubt, innerhalb Methoden einer Klasse, auf das aktuelle Objekt zuzugreifen.

Konstruktoren - Anwendung

Erstellen einer `Koordinate` ist relativ umständlich:

```
Koordinate k = new Koordinate();
k.breitengrad = 45.97f
k.laengengrad = 7.65f
```

Konstruktoren - Anwendung

Erstellen einer `Koordinate` ist relativ umständlich:

```
Koordinate k = new Koordinate();
k.breitengrad = 45.97f
k.laengengrad = 7.65f
```

Konstruktoren erlauben es, einfacher die Initialwerte der Felder einer Klasse zu setzen!

```
Koordinate k = new Koordinate(45.97f, 7.65f);
```

Konstruktoren - Definition

```
public class Koordinate{
    float breitengrad;
    float laengengrad;

    // Konstruktor fuer gegebene Koordinaten
    Koordinate(float b, float l){
        this.breitengrad = b;
        this.laengengrad = l;
    }

    // Standard-Konstruktor ohne Parameter
    Koordinate(){}
}
```

Statische Felder und Methoden

Mit dem Schlüsselwort `static` deklariert.

- Existieren nur einmal pro Klasse.
- Werden direkt über die Klasse aufgerufen, nicht auf Objekten der Klasse.
- Beobachtung: die `main` Methode ist statisch!
  `public static void main(String[] args)`

Beispiel: Die `Math` Klasse

```
float f = Math.sqrt(g)
```

Beispiel: Die `Math` Klasse

```
float f = Math.sqrt(g)
```

Ist definiert in der Klasse `Math`:

```
public class Math {
    public static float sqrt(float value){
        // ...
    }

    // ...
}
```

Beispiel: Die `Math` Klasse

```
float f = Math.sqrt(g)
```

Ist definiert in der Klasse `Math`:

```
public class Math {
    public static float sqrt(float value){
        // ...
    }

    // ...
}
```

Beispiel zum Konzept der Datenkapselung

Szenario / Motivation:

- Langfristiges Ziel: Netzberechnung für die Wasserversorgung
- Komponenten: Leitungen, Reservoire, *Pumpen*
- Modellierung der Komponenten durch Objekte (Klassen)

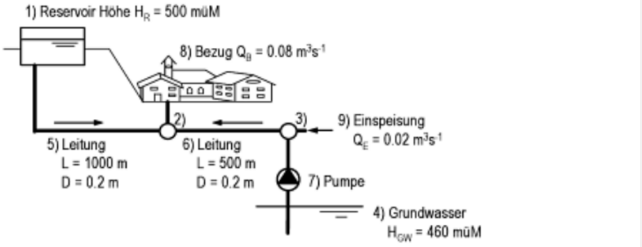


Abb. 11.12. Beispiel zur Netzberechnung: Ein einfaches Wasserversorgungsnetz

(Quelle: Willi Gujer, Siedlungswasserwirtschaft, Springer)

Pumpe

```
public class Pumpe{
  public double h; // manometrische Foerderhoehe in m
  public double q; // Foerdermenge in m^3/s

  public Pumpe(double H, double Q){
        h = H; q = Q;
        assert(h > 0); assert(q > 0); // Invariante!
  }
}
 ...
Pumpe p1 = new Pumpe(40, 0.02);
Pumpe p2 = new Pumpe(30, 0.01);
```

Szenario

Jemand verwendet die Pumpendaten

```
double v = p1.q * 20;   // Foerdermenge in 20 Sekunden
```

Szenario

Jemand verwendet die Pumpendaten

```
double v = p1.q * 20;   // Foerdermenge in 20 Sekunden
```

und verändert die Parameter der Pumpe

```
p1.h = 50; p1.q = 0;
```

Das wollen wir nicht! Wie vermeidet man diese Inkonsistenz?

Kapselung = Information Hiding!

```
public class Pumpe{
  private double h;
  private double q;

  public Pumpe(double H, double Q){
        h = H; q = Q;
        assert(h > 0); assert(q > 0); // Invariante!
  }

  public double getQ(){
        return q;
  }
}
```

Kapselung = Information Hiding!

```
public class Pumpe{
  private double h;
  private double q;

  public Pumpe(double H, double Q){
        h = H; q = Q;
        assert(h > 0); assert(q > 0); // Invariante!
  }

  public double getQ(){
        return q;
  }
}
```

Felder sind nun versteckt. Zugriff nur innerhalb der Klasse.

Getter Methode: erlaubt (lesenden) Zugriff auf q

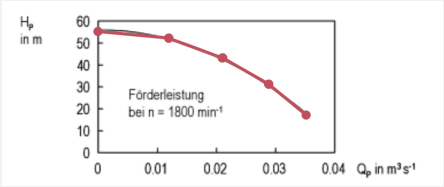
Datenkapselung

- Eine komplexe Funktionalität wird auf einer möglichst hohen Abstraktionsebene semantisch definiert und durch ein vereinbartes minimales *Interface* zugänglich gemacht
- *Wie* der Zustand durch die Datenfelder einer Klasse repräsentiert wird, sollte für den Benutzer der Klasse nicht sichtbar sein
- Dem Benutzer der Klasse wird eine *repräsentationsunabhängige* Funktionalität angeboten
- Dies gestattet eine garantierte Einhaltung von *Invarianten*.[9]

[9] Im Beispiel: $q \neq 0$ und $h \neq 0$

Klassen können mehr als Records

```
public class Pumpe{
  private double[] ha; // Kennlinie
  private double[] qa;

  public Pumpe(double[] Ha,double []Qa){
        ha = Ha; qa = Qa;
  }

  private interpolate(double x, double X[], double Y[]) {...}

  public double getQ(double h){
        return interpolate(h,Ha,Qa);
  }
}
```



Verwendung

```
double[] kennlinieH = {55,50,40,30,20};
double[] kennlinieQ =  {0,0.01,0.02,0.03,0.035};
Pumpe p = new Pumpe(kennlinieH, kennlinieQ);

// Foerdermengen in 1 Stunde bei 45 m Foerderhoehe
double v = p.GetQ(45) * 3600;
```

Klassen

- Klassen beherbergen das Konzept der *Datenkapselung.*
- Klassen sind Bestandteil nahezu jeder objektorientierten Programmiersprache.
- Klassen können in Java in Paketen zusammengefasst werden.

Zugriffsmodifizierer

```
public class Pumpe{
  private double h;
  private double q;
   ..
}
```

Sichtbarkeit nach Modifizierer:

	Klasse	Paket	Subklasse	Global
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	
(keines)	✓	✓		
`private`	✓			