

Lernziele

- Sie können eigene *Klassen/Datentypen* erstellen.
- Sie verstehen, wie *Objekte* von Klassen instanziiert und verwendet werden.
- Sie kennen den Begriff der *Datenkapselung* und können dies anwenden.

380

Definition: *Klassen*

Klassen sind (selbstdefinierte) Datentypen, die es erlauben, mehrere Elemente zu einem neuen Objekt zusammenzufassen und mit einem gemeinsamen Namen anzusprechen.

Buch auf Seite 129

382

13. Java Klassen

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

381

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 (typischerweise) separate `.java` Dateien mit gleichem Namen



383

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.

Beispiel: Erdbebendaten



SED > Earthquake catalog > Query the catalogue

Earthquake catalog

link	date	time	appraisal	event type	lat [°N]	lon [°E]	source agency	depth	Mw	MI	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-09)	5.	1.94	1.6			

384

385

Klasse für Messwert - Erster Versuch

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

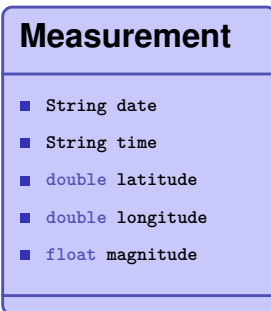
Datei Measurement.java:

```
public class Measurement {

    String date;
    String time;

    double latitude;
    double longitude;

    float magnitude;
}
```



Definition: *Objekte*

Klassen sind Datentypen. Objekte sind die Werte eines solchen Typs, wobei die Klasse die Struktur ihrer Objekte vorgibt.

Buch auf Seite 130

386

387

Objekte: Instanzen von Klassen

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

⇒ Vergleichbar mit den *Kopfdaten* aus dem CSV.

Objekte werden instanziiert nach Bauplan und enthalten nun Werte.

⇒ Vergleichbar mit den einzelnen *Datenzeilen* aus dem CSV.

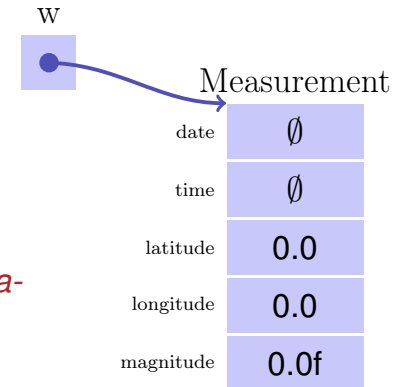
Objektinstanziierung: Das Schlüsselwort `new`

Variable "w" vom Typ "Measurement"

```
Measurement w;
```

```
w = new Measurement();
```

Instanziierung eines Objekts vom Typ Measurement



388

389

Dereferenzierung: Zugriff auf Felder

```
Measurement w;
```

```
w = new Measurement();
```

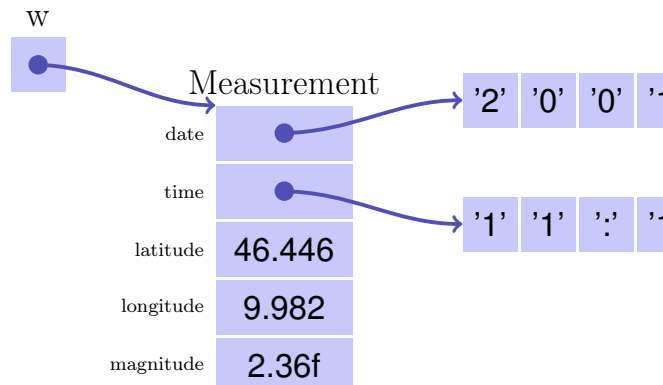
```
w.date = "2001/01/03";
```

```
w.time = "11:11:20";
```

```
w.latitude = 46.446;
```

```
w.longitude = 9.982;
```

```
w.magnitude = 2.36f;
```



390

Objekte sind Referenztypen: Aliasing

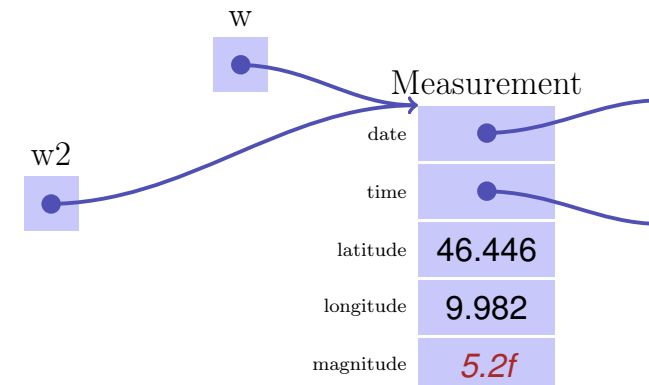
```
Measurement w;
```

```
w = new Measurement();
```

```
Measurement w2 = w;
```

```
w2.magnitude = 5.2f;
```

```
Out.println(w.magnitude);
```

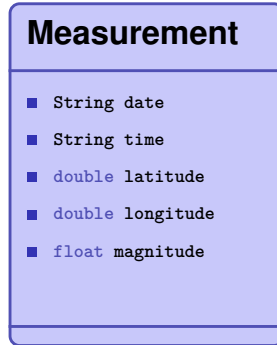


391

Moment mal! ...

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

Gutes Klassendesign?



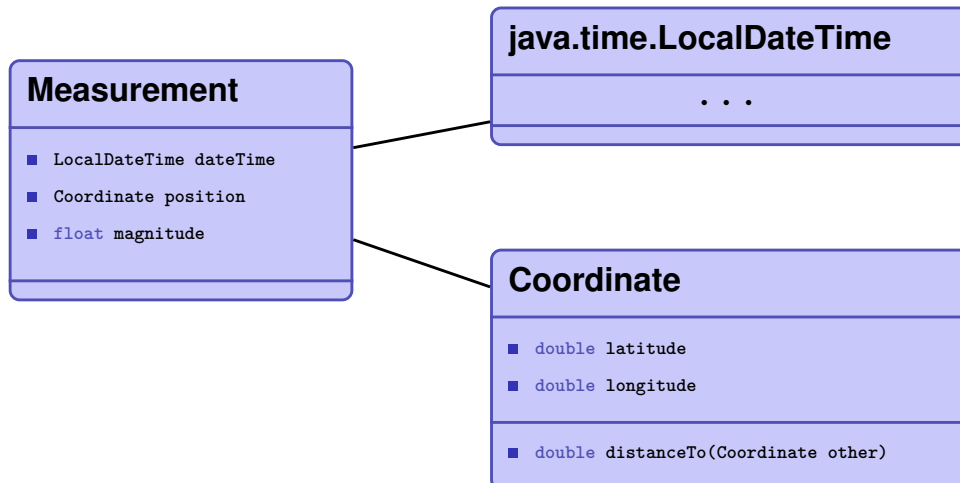
Das können wir besser!

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

392

393

Klassendesign - zweiter Versuch



Methoden in Klassen

```
public class Coordinate {

    double latitude;
    double longitude;

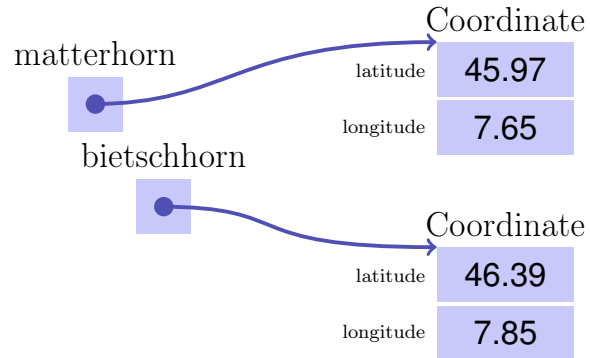
    /**
     * Computes the distance to the provided coordinate 'other'.
     */
    double distanceTo(Coordinate other){
        double dl = this.latitude - other.latitude;
        // complete this as exercise ...
    }
}
```

394

395

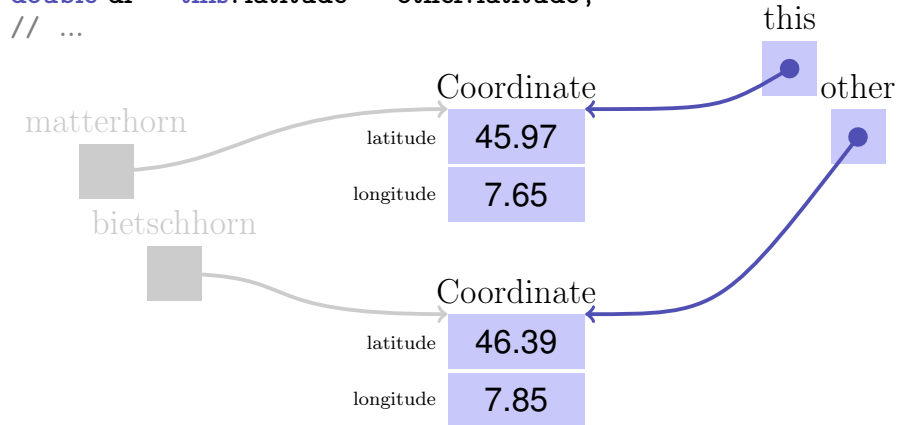
Methodenaufwurf - Beispielsetting

```
Coordinate matterhorn, bietschhorn;  
// ... Instanciate and set values ...  
d = matterhorn.distanceTo(bietschhorn);
```



Im Kontext innerhalb der Methode

```
double distanceTo(Coordinate other){  
    double dl = this.latitude - other.latitude;  
    // ...
```



396

397

Schlüsselwort `this`

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

Konstruktoren

Erstellen einer `Coordinate` ist relativ umständlich:

```
Coordinate k = new Coordinate();  
k.latitude = 45.97;  
k.longitude = 7.65;
```

Konstruktoren erlauben es, einfacher die Initialwerte der Felder eines neu erstellten Objektes zu setzen.

```
Coordinate k = new Coordinate(45.97, 7.65);
```

Generell ist die Funktion des Konstruktors, das Objekt in einen sinnvollen "gültigen" Zustand zu bringen.

398

399

Konstruktoren - Definition

```
public class Coordinate{
    double latitude;
    double longitude;

    // Constructor for a given coordinates (as a pair of lat/long).
    Coordinate (double lat, double lon){
        this.latitude = lat;
        this.longitude = lon;
    }
}
```

400

Definition: *Datenkapselung*

Mittels Datenkapselung kann der Zugriff von ausserhalb der Klasse auf Daten und Code einer Klasse gesteuert werden.

Buch auf Seite 246

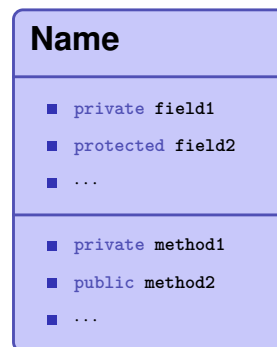
401

Datenkapselung / Information Hiding

Steuern, welche Daten und welcher Code woher *zugänglich* ist.

Zugriffsmodifikatoren:

- **private**: Sichtbar aus Code derselben Klasse
- **protected**: Sichtbar aus Code derselben Klasse oder Unterklasse (später)
- **public**: Von überall sichtbar



402

Beispiel: Koordinate

```
public class Coordinate {
    public double latitude;
    public double longitude;

    public double distanceTo(Coordinate other){...}
}
```

Probleme:

- Ungültige Wertzuweisungen möglich
- Konsistenzprüfungen nicht möglich
- Implementierung exponiert

403

Koordinate: Zugriffsmethoden

```
public class Coordinate {
    private double latitude;
    private double longitude;

    public double getLatitude(){
        return latitude;
    }

    public void setLatitude(double lat){
        assert lat >= -90 && lat <= 90;
        this.latitude = lat;
    }
    //...
```

404

Koordinate: Benützung

```
Coordinate position = ...;
position.setLatitude(45);    //This is fine

Out.println(position.getLatitude());    //This is fine

// The following two lines are WRONG
position.setLatitude(100);    //Assertion violation at runtime
Out.print(position.latitude); //Doesn't compile. Invalid access
```

405

Kapselung: Implementierung austauschen

Ohne direkten Zugriff auf Daten kann Implementierung einfach ausgetauscht werden, ohne dass dies "nach aussen" sichtbar wird.

406

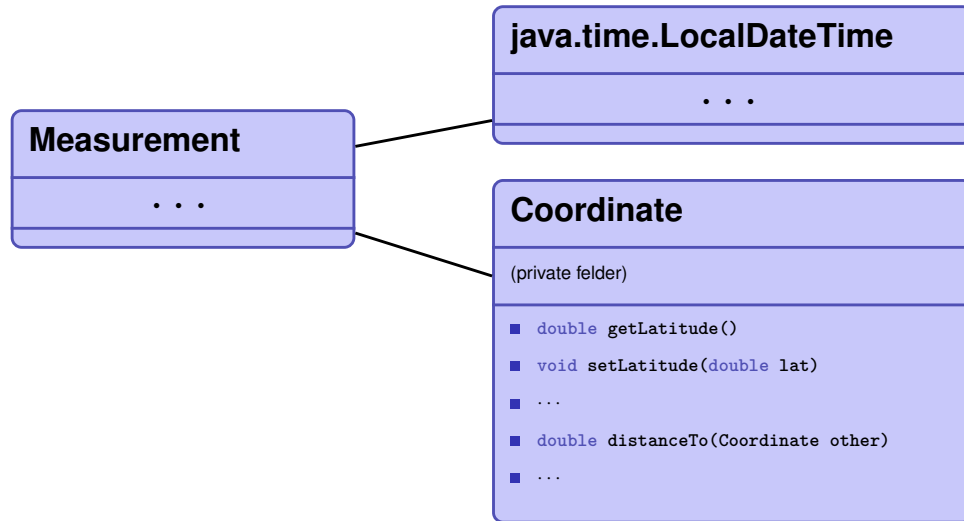
Beispiel: Wechsel zu Schweizer Koordinaten

```
public class Coordinate {
    // Coordinate in LV03 Format (Swiss coordinate grid)
    private int x;
    private int y;

    public double getLatitude(){
        double x_aux = (x - 200_000) / 1_000_000;
        double y_aux = (y - 600_000) / 1_000_000;
        double result = (16.9023892 + (3.238272 * x_aux))
            - (0.270978 * pow(y_aux, 2)) - (0.002528 * pow(x_aux, 2))
            - (0.0447 * pow(y_aux, 2) * x_aux) - (0.0140 * pow(x_aux, 3));
        return (result * 100) / 36;
    }
}
```

408

Klassendesign - dritter Versuch



409

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*

410

Definition: *Statische Felder und Methoden*

Statische Methoden und Felder werden nicht pro Objekt angelegt, sondern nur einmal pro Klasse. Der Zugriff geschieht direkt ueber die Klasse.

Buch auf Seite 151

411

Statische Felder und Methoden

Mit dem Schlüsselwort `static` deklariert.

- Existieren nur einmal pro Klasse.
- Werden direkt über den Klassennamen aufgerufen, nicht auf Objekten der Klasse...
- ...deswegen kann aus statischen Methoden nicht auf `this` zugegriffen werden.
- Beobachtung: die `main` Methode ist statisch!
`public static void main(String[] args)`

412

Beispiel: Die In Klasse

```
int f = In.readInt()
```

Ist definiert in der Klasse In (nächste Folie)

Beispiel: Die In Klasse

```
/** This method skips white space and tries to read an integer. If the
    text does not contain an integer or if the number is too big, the
    value 0 is returned and the subsequent call of done() yields false.
    An integer is a sequence of digits, possibly preceded by '-'.
    */
public static int readInt(){
    String s = readDigits(); // read as many digits as possible
    try {
        done = true;
        return Integer.parseInt(s); // trt to interpret string s as int
    } catch (Exception e) {
        done = false;
        return 0; // something other than digits reat, return 0 instead
    }
}
```