Educational Objectives

- You can create your own classes/data types.
- You understand how objects are being instantiated and used.
- You know the term encapsulation and are able to apply it to your situation.

13. Java Classes

Classes, types, objects, declaration, instantiation, constructors, encapsulation, static fields and methods

Definition: Classes

Classes are (user-defined) data types that allow to combine several elements to a new object and to access it by a common name.

A class is an entity with a name that contains data and functionality.

- A class defines a new data type.
- Data consists of variables that we call fields or attributes.
- Functionality consists of methods that are defined within the class.
- Classes are (typically) separate .java files with the same name.

Classes - Technical

- field1
- field2
- ...
- method1
- method2
- ...

Book on page 129
Classes facilitate to *bundle* the data that *belongs together* content wise.

Classes provide *functionality* that allows to perform *queries* based on the data or *operations* on the data.

### Example: Earthquake catalog

#### Definition: Objects

Classes are data types. Objects are values of such a type, where the class determines the structure of those objects.

#### Class for measurement - first try

```java
public class Measurement {
    String date;
    String time;
    double latitude;
    double longitude;
    float magnitude;
}
```

#### File Measurement.java

```java
public class Measurement {
    String date;
    String time;
    double latitude;
    double longitude;
    float magnitude;
}
```
**Objects: Instances of Classes**

*Classes* describe the structure of objects, like a *blueprint*

⇒ Comparable with the *header* of the CSV.

*Objects* are instantiated according to the blueprint and will contain values

⇒ Comparable with the individual *data-rows* in the CSV.

**Object Instantiation: The Keyword `new`**

```
Variable “w” of type “Measurement”

Measurement w;
w = new Measurement();

Instantiation of an object of type Measurement

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>()</td>
</tr>
<tr>
<td>time</td>
<td>()</td>
</tr>
<tr>
<td>latitude</td>
<td>0.0</td>
</tr>
<tr>
<td>longitude</td>
<td>0.0</td>
</tr>
<tr>
<td>magnitude</td>
<td>0.0f</td>
</tr>
</tbody>
</table>
```

**De-referencing: Accessing Fields**

```
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;

De-referencing: “Follow the arrow”

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

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

w.latitude = 46.446;
w.longitude = 9.982;
w.magnitude = 2.36f;
```

**Objects are Reference-Types: Aliasing**

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

w2 = w;

w2.magnitude = 5.2f;

println(w.magnitude);
```

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

w2 = w;
w2.magnitude = 5.2f;

println(w.magnitude);
```
Wait a second! ...

Classes facilitate to **bundle** the data that **belongs together** content wise.

**Good Class Design?**

- Date and Time belong together in a separate class: Java already offers this: `java.time.LocalDateTime`
- Latitude and longitude belong in their own data type `Coordinate`.

**Class Design - second try**

- Measurement
  - String date
  - String time
  - double latitude
  - double longitude
  - float magnitude

- Coordinate
  - double latitude
  - double longitude
  - double distanceTo(Coordinate other)

**Methods in Classes**

```java
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 ...
    }
}
```
**Method calls - Example setup**

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

**From the context inside the method**

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

**Keyword this**

The `this` keyword enables to access the current object from within a method of that class.

**Constructors**

Creating a `Coordinate` is somewhat cumbersome:

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

Constructors facilitate to easily set the initial values of a newly created object.

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

In general, the job of the constructor is to establish a reasonable “valid” state.
**Constructors - Definition**

```java
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;
    }
}
```

**Definition: Data Encapsulation**

Data encapsulation allows to control access from outside to data and code of the class.

Book on page 246

---

**Data Encapsulation / Information Hiding**

Control, what data and what code can be *accessed* from where.

Access modifiers:

- **private**: Visible only from code within the same class
- **protected**: Visible from code in the same class or a subclass (later)
- **public**: Visible from everywhere

---

**Example: Coordinate**

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

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

Problems:

- Assignment of invalid values
- Consistency checks not possible
- Implementation exposed
Coordinate: Accessor Methods

```java
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;
    }

    //...
}
```

Coordinate: Usage

```java
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
```

Encapsulation: Exchange implementation

With no direct access to the data, it is easy to change the implementation without making it visible “to the outside”.

Example: Switch to Swiss Coordinate Grid

```java
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;
    }
}
```
Class Design - third try

Measurement

```
java.time.LocalDateTime
```

Coordinate

(private fields)

```
double getLatitude()
void setLatitude(double lat)
· · ·
double distanceTo(Coordinate other)
· · ·
```

Data Encapsulation

- A complex functionality gets defined as abstract as possible semantically and made accessible trough an agreed-upon minimal interface
- It should not be visible for the client how the state is represented in data fields of the class
- The class provides functionality to the client independently of its representation
- This allows to enforce invariants

Definition: Static Fields and Methods

Static methods and fields are not instantiated per object, but only once per class. They can be accessed directly via the class.

Book on page 151

Static Fields and Methods

- Declared with the keyword `static`.
- Exist only once per class
- Are accessed directly via the class name rather than objects of the class...
- ...this is why it’s not possible to access `this` from static methods.
- Observation: the `main` method is static!
  ```java
  public static void main(String[] args)
  ```
Example: The In class

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

Is defined in class In (next slide)

/*
** 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 '−'.
*/

```java
public static int readInt(){
    String s = readDigits(); // read as many digits as possible
    try {
        done = true;
        return Integer.parseInt(s); // try to interpret string s as int
    } catch (Exception e) {
        done = false;
        return 0; // something other than digits read, return 0 instead
    }
}
```