1 Algorithmus

Definition: **Endliches**, **deterministisches** und **allgemeines** Verfahren unter Verwendung **ausführbarer**, **elementarer** Schritte.

2 Input und Output

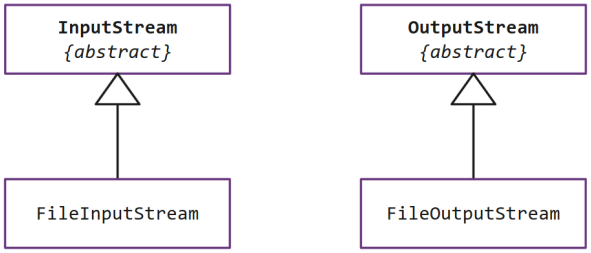

Abbildung 1: Klassenhierarchie von Input und Output

2.1 Input

2.1.1 File-Reader

```
try (var reader = new FileReader("quotes.txt")) {
  int value = reader.read();
  while (value ⩾ 0) {
    char c = (char) value;
    // use character
    value = reader.read();
  }
}
```

```
new FileReader(f);
// ist äquivalent zu
new InputStreamReader(new FileInputStream(f));
```

2.1.2 Zeilenweises Lesen

```
try (var reader = new BufferedReader(new FileReader("quotes.txt")) {
  String line = reader.readLine();
  while (line != null) {
    System.out.println(line);
    line = reader.readLine();
  }
}
```

Info: `FileReader` liest einzelne Zeichen, `BufferedReader` liest ganze Zeilen.

2.2 Output

2.2.1 File-Writer

```
try (var writer = new FileWriter("test.txt", true)) {
  writer.write("Hello!");
  writer.write("\n");
}
```

2.3 Zusammenfassung

- Byte-Stream: Byteweises Lesen von Dateien
  - FileInputStream, FileOutputStream
- Character-Stream: Zeichenweises Lesen von Dateien (UTF-8)
  - FileReader, FileWriter

3 Serialisierung

Das `Serializable`-Interface implementieren (Marker-Interface). Ohne Marker-Interface wird eine `NotSerializableException` geworfen. Jedes Feld, das serialisiert werden soll, muss ebenfalls `Serializable` implementieren (Transitive Serialisierung).

```
class Person implements Serializable {
  private static final long serialVersionUID = 1L;
  private String firstName;
  private String lastName;
  // …
}
```

Das kann dann vom ObjectOutputStream verwendet werden, um Data Binär zu serialisieren:

```
try (var stream = new ObjectOutputStream(new
FileOutputStream("serial.bin"))) {
  stream.writeObject(person);
}
```

Um ein Objekt aus einem Bytestrom zu deserialisieren, wird der ObjectInputStream verwendet:

```
try (var stream = new ObjectInputStream(
  new FileInputStream("serial.bin"))) {
  Person p = (Person) stream.readObject();
  // …
}
```

3.1 Serialisierung mit Jackson

```
Employee e = new Employee(1, "Frieder Loch");
String jsonString = mapper.writeValueAsString(e);
var writer = new PrintWriter(FILE_PATH);
writer.println(jsonString);
writer.close();
```

Output:

```
{"id":1,"name":"Frieder Loch"}
```

3.1.1 Beeinflussung der Serialisierung

```
public class WeatherData {
  @JsonProperty("temp_celsius")
  private double tempCelsius;
}
```

```
@JsonPropertyOrder({"name", "id"})
public class Employee{
  public int id;
  public String name;
}
```

`@JsonIgnore`, `@JsonInclude(Include.NON_NULL)` (nur nicht-null-Werte), `@JsonFormat(pattern = "dd-MM-yyyy")`

```
@JsonRootName(value="user")
public class Customer {
  public int id;
  public String name;
}
// ...
var mapper = new ObjectMapper().enable(
  SerializationFeature.WRAP_ROOT_VALUE
);
```

Output:

```
{
  "user": {
    "id": 1,
    "name": "Frieder Loch"
  }
}
```

3.1.2 JsonGenerator

```
var generator = new JsonFactory().createGenerator(
  new FileOutputStream("employee.json"), JsonEncoding.UTF8);
  jsonGenerator.writeStartObject();
  jsonGenerator.writeFieldName("identity");
  jsonGenerator.writeStartObject();
  jsonGenerator.writeStringField("name", company.name);
  jsonGenerator.writeEndObject();
```

3.1.3 Deserialisierung

```
String json = "{\"name\":\"Max\", \"alter\":30}";
ObjectMapper mapper = new ObjectMapper();
Benutzer benutzer = mapper.readValue(json, Benutzer.class); // throws
JsonMappingException
```

Deserializer:

```
public class CompanyJsonDeserializer extends JsonDeserializer {
  @Override
  public Company deserialize(JsonParser jP, DeserializationContext dC)
throws IOException {
    var tree = jP.readValueAs(JsonNode.class);
    var identity = tree.get("identity");
    var url = new URL(tree.get("website").asText());
    var nameString = identity.get("name").asText();
    var uuid = UUID.fromString((identity.get("id").asText()));
    return new Company(nameString, url, uuid);
  }
}
```

@JacksonInject:

```
public class Book {
  public String name;
  @JacksonInject
  public LocalDateTime lastUpdate;
}
InjectableValues inject = new InjectableValues.Std()
  .addValue(LocalDateTime.class, LocalDateTime.now());
Book[] books = new ObjectMapper().reader(inject)
  .forType(new TypeReference<Book[]>(){}).readValue(jsonString);
```