JAVA 07

Persistenz mit JSON & Files

JAVA 07 - Persistenz mit JSON & Files

1. **FileWriter – In Dateien schreiben**
2. **FileReader – Aus Dateien lesen**
3. **Was ist „Jackson2“?**
4. **JAVA-Model ⇔ JSON ⇔ Datei**
5. **Anleitung - Jackson2 mit IntelliJ nutzen**

1. FileWriter – In Dateien schreiben

```
String textToWrite = „Diesen Text möchten wir speichern!“
File file = new File(„example.txt“);
FileWriter writer = null;
try {
        file.createNewFile();           // wenn es die Datei schon gibt, passiert nichts
        writer = new FileWriter(file);  // einen „Writer“ für diese Datei
        writer.write(input);            // in den Speicher (!) schreiben
        writer.flush();                 // Speicherinhalt in die Datei (Datei speichern)
} catch (IOException e) {  e.printStackTrace();
} finally {
        if (writer != null) {
                try { writer.close();
                } catch (IOException e) { e.printStackTrace(); }
        }
}
```

2. FileReader – Aus Dateien lesen

```
String content = null;
File file = new File(„example.txt“);
FileReader reader = null;
try {
        reader = new FileReader(file);     // einen „Reader“ für diese Datei
        // über das File-Objekt bekommen wir die Größe des Inhalts heraus!
        char[] chars = new char[(int) file.length()];       // wir können nur characters lesen
        reader.read(chars);                 // die einzelne Zeichen lesen. *IN den Parameter*
        content = new String(chars);        // einen String aus dem character-Array erstellen
} catch (IOException e) { e.printStackTrace();
} finally {
        if(reader !=null) {
                try { reader.close();
                } catch (IOException e) { e.printStackTrace(); }
        }
}
```

3. Was ist Jackson2?

Oder: Warum wir Dateien NICHT „zu Fuß“ laden und speichern:

- Dateien mit den gegebenen Standards von JAVA manuell zu speichern und zu laden ist relativ aufwendig. Ein wichtiger Punkt ist auch, dass wir zunächst Daten in JSON umwandeln müssten.

- Stattdessen nutzen wir das Wissen anderer Entwickler,

  die sich ausgiebig mit **wiederkehrenden Problemen** befasst haben

- Im Fall der Kombination von **Files** und **JSON** ist **Jackson** eine bewerte **Library**

- Die von uns verwendete Version ist **Jackson 2** (Wird seit **2012** fortwährend weiter entwickelt)

4. JAVA-Model ⇔ JSON ⇔ Datei

```
Person p1 = new Person();
Person p2 = new Person();
Person p3 = new Person();

P1.setName("John");
P2.setName("Mary");
P3.setName("Peter");

P1.setCash(1000);
P2.setCash(2350);
P3.setCash(700);

Person[] pArr = { p1, p2, p3 };
```

JSON-Darstellung des Personenarrays „pArr“

```
{[
    {   "Name":"John",
        "cash":1000}
    {   "Name":"Mary",
        "cash":2350}
    {   "Name":"Peter",
        "cash":700}
]}
```

4. JAVA-Model ⇔ JSON ⇔ Datei

2.2. Abspeichern: Objekt > JSON > Datei

```
ObjectMapper mapper = new ObjectMapper();
Person person = new Person(„Max Musterman");
try {
        // Example 1: Convert object to JSON string and pretty print
        jsonInString = mapper.writerWithDefaultPrettyPrinter().writeValueAsString(person);
        System.out.println(jsonInString);

        // Example 2: Convert object to JSON string and save into a file directly
        mapper.writeValue(new File("D:\\personTest.json"), person);

} catch (Exception e) {
        e.printStackTrace();
}
```

4. JAVA-Model ⇔ JSON ⇔ Datei

2.3. Laden: Date > JSON > Objekt

```
ObjectMapper mapper = new ObjectMapper();
Person person = null;
try {

        // Convert JSON string from file to Object
        person = mapper.readValue(new File("D:\\personTest.json"), Person.class);

} catch (Exception e) {
        e.printStackTrace();
}
```

4. JAVA-Model ⇔ JSON ⇔ Datei

2.4. Die Persistenz-Klassen

- Zu den Klassen des MVC können wir nun eine weitere Kategorie hinzufügen, die zum speichern und laden von Objekten zuständig sind
- Diese Prozeduren gehört zum **Controller** Anteil, wird aber eher bei den **services** untergebracht

```
public class PersonPersistence {

        public static void store(Person person) {
                // Abspeichern der Person in einer Datei mittels Methode unter 2.2
        }

        public static Person load() {
                // Laden der Person aus einer Datei mittels Methode unter 2.3
        }

}
```

5. Anleitung - Jackson2 mit IntelliJ nutzen

Damit Sie Jackson wie in diesem Dokument beschrieben verwenden können, müssen Sie die Libraries aus dem Netz herunterladen. Navigieren Sie unter **IntelliJ** zu folgendem Menüpunkt:

-> File

-> Project Structure

-> Project Settings

-> Libraries

Klicken Sie auf das grüne **PLUS** (**New Project Library**) und wählen Sie **From Maven** und geben Sie folgenden Suchbegriff ein:

com.fasterxml.jackson.core:jackson-databind

(es reicht auch der Suchbegriff „Jackson“ und aus der erscheinenden Auswahl heraussuchen)

Hier wählen Sie die neueste Version (Bei Erstellung dieses Dokuments 2.8.5) und bestätigen Sie die **Checkboxen** für „**Download to**“, „**Sources**“ und „**JavaDocs**“. Bestätigen Sie die Auswahl mit OK und warten Sie bis es heruntergeladen wird. Bestätigen Sie, dass die Library zum Projekt hinzugefügt werden soll.

Maven ist für JAVA eine öffentliche Stelle für Libraries, mit denen alle möglichen Problemstellungen von Entwicklern weltweit bereit gestellt wird. Generell gilt hier das Prinzip des OpenSource, also können die hier verwendeten Lösungen auch kommerziell frei verwendet werden.