Name:	
Klasse:	
Datum:	

Schuljahr: 2024/25

Apache Maven gehört zu den Build-Tools, wie beispielsweise Apache Ant, die in Java entwickelt worden sind. Inzwischen hat sich Maven zum kompletten Projektmanagement-Werkzeug mit vielen Erweiterungen (Plugins) entwickelt. Wir benötigen Maven im ersten Schritt, um Frameworks bzw. Bibliotheken und deren Abhängigkeiten (Dependency Management) zu verwalten. Maven ist in Eclipse integriert, kann aber auch unabhängig davon verwendet werden.

Was die KI (Bing) zu Apache Maven schreibt

„Apache Maven ist ein leistungsstarkes Werkzeug für das Projektmanagement und die Automatisierung von Build-Prozessen, insbesondere für Java-basierte Projekte. Zu den Hauptvorteilen von Maven gehört die Konvention über Konfiguration, was bedeutet, dass Entwickler weniger Zeit mit der Einrichtung ihres Projekts verbringen müssen, da Maven viele Standards vorgibt, die allgemein anerkannt sind. Dies führt zu einer konsistenten und leicht verständlichen Projektstruktur. Maven verwaltet Abhängigkeiten effizient, indem es automatisch alle benötigten Bibliotheken und Plugins herunterlädt und in das Projekt integriert. Es unterstützt auch die Wiederverwendung von Code durch seine zentrale Repository-Verwaltung, was die Konsistenz über mehrere Projekte hinweg fördert. Darüber hinaus erleichtert Maven die Erstellung von Berichten, die Ausführung von Tests und die Generierung von Dokumentationen, was zu einer verbesserten Qualität und Wartbarkeit des Codes beiträgt. Die Integration in gängige Entwicklungsumgebungen und die Plattformunabhängigkeit machen Maven zu einem unverzichtbaren Werkzeug in modernen Entwicklungsprozessen.“

1 Voraussetzungen

- OpenJDK-21 LTS
- Maven 3.9.6 oder neuer
- Eclipse IDE for Enterprise Java and Web Developers - 2024-09 oder neuer

Versionstest in der Eingabeaufforderung / Konsole:

unter Windows:

```
E:\progs>java -version
openjdk version "21.0.2" 2024-01-16
OpenJDK Runtime Environment (build 21.0.2+13-58)
OpenJDK 64-Bit Server VM (build 21.0.2+13-58, mixed mode, sharing)

E:\progs>mvn -version
Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Maven home: E:\progs\apache-maven
Java version: 21.0.2, vendor: Oracle Corporation, runtime: E:\progs\jdk21
Default locale: de_DE, platform encoding: UTF-8
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

unter Linux:

```
mgn@mars:~$ java -version
openjdk version "21.0.2" 2024-01-16
OpenJDK Runtime Environment (build 21.0.2+13-Ubuntu-122.04.1)
OpenJDK 64-Bit Server VM (build 21.0.2+13-Ubuntu-122.04.1, mixed mode, sharing)

mgn@mars:~$ mvn -version
Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Maven home: /home/mgn/progs/apache-maven
Java version: 21.0.2, vendor: Private Build, runtime: /usr/lib/jvm/java-21-openjdk-amd64
Default locale: de_DE, platform encoding: UTF-8
OS name: "linux", version: "6.5.0-26-generic", arch: "amd64", family: "unix"
```

Notiz:

https://it.gentiana-clusii.de/ CC BY-NC-ND

2 Das Herz von Maven – die pom.xml Datei

Die Basisdatei für Maven hat den Namen `pom.xml`.

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.
     xsd">
4    <modelVersion>4.0.0</modelVersion>
5
6    <groupId>de.gc.mav</groupId>
7    <artifactId>TabellenTest</artifactId>
8    <version>0.0.1</version>
```

In Zeile 1..4 findet sich der Header der XML-Datei, der u. a. festlegt, welche Maven-Modelversion verwendet wird.

In Zeile 6..8 wird das Projekt definiert:

`groupId` Hier wird normalerweise der Namespace angegeben.

`artifactId` Hier wird der Name des Projekts angegeben.

`version` Die Versionsnummer.

```
10   <properties>
11      <maven.compiler.source>21</maven.compiler.source>
12      <maven.compiler.target>21</maven.compiler.target>
13      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
14
15      <maven.compiler.plugin.version>3.12.1</maven.compiler.plugin.version>
16      <maven.antrun.plugin.version>3.0.0</maven.antrun.plugin.version>
17
18      <main.class>de.gc.mav.SimpleTable</main.class>
19   </properties>
```

Im Bereich `properties` werden die Grundeinstellungen definiert.

`maven.compiler.source`	Welche Java-Version soll für den Sourcecode verwendet werden.
`maven.compiler.target`	Für welche Java-Version soll übersetzt werden.
`project.build.sourceEncoding`	Welches Encoding wird für die Javadateien verwendet.
`maven.compiler.plugin.version`	Welche Version soll für das Maven-Compiler-Plugin verwendet werden.
`maven.antrun.plugin.version`	Welche Version soll für das Maven-AntRun-Plugin verwendet werden.
`main.class`	Welche Klasse soll für den Start aus Maven verwendet werden.

```
21   <dependencies>
22      <dependency>
23         <groupId>org.nocrala.tools.texttablefmt</groupId>
24         <artifactId>text-table-formatter</artifactId>
25         <version>1.2.4</version>
26      </dependency>
27   </dependencies>
```

Will man externe Bibliotheken verwenden, wird oft für diese Bibliotheken die Abhängigkeit angegeben, z. B. für den „Text Table Formatter“. Über das Maven-Repository findet man schnell die gewünschten Bibliotheken bzw. das Frameworks.

Diese Einträge kopiert man dann in die `pom.xml`-Datei in den Bereich `dependencies`.

```
29    <build>
30       <defaultGoal>clean compile</defaultGoal>
31
32       <plugins>
33
34          <!-- create directories -->
35          <!-- mvn antrun:run@createFolder -->
36          <plugin>
37             <groupId>org.apache.maven.plugins</groupId>
38             <artifactId>maven-antrun-plugin</artifactId>
39             <version>${maven.antrun.plugin.version}</version>
40             <executions>
41                <execution>
42                   <id>createFolder</id>
43                   <phase>generate-dirs</phase>
44                   <configuration>
45                      <target>
46                         <mkdir dir="./src/main/java" />
47                         <mkdir dir="./src/main/resources" />
48                         <mkdir dir="./src/test/java" />
49                         <mkdir dir="./src/test/resources" />
50                         <mkdir dir="./target" />
51                      </target>
52                   </configuration>
53                   <goals>
54                      <goal>run</goal>
55                   </goals>
56                </execution>
57             </executions>
58          </plugin>
```

Mit Plugins kann Maven erweitert werden. Einige Plugins werden mit der Standardkonfiguration automatisch eingebunden, z. B. das Compiler-Plugin. Will man hier aber andere Einstellungen verwenden, so muss es als „Plugin“ eingebunden werden. Diese werden in dem Bereich `build-plugins` definiert. Dabei werden die Default-Vorgänge in `defaultGoal` definiert, hier löscht man zuerst alle erstellten Dateien (z. B. class-Dateien) und dann ruft man den Compiler auf.

Das Plugin `maven-antrun-plugin` dient dazu, Apache-ANT-Task auszuführen, hier konkret, um die notwendigen Verzeichnisse zu erstellen.

Die Standarddateistruktur bei Maven sieht meist wie folgt aus:

- Bereich `src`
  - `main` Hier werden alle Dateien für das Programm gespeichert.
    - `java` java-Dateien
    - `resources` Bilder, Icons, ...
  - `test` Hier werden alle Dateien für das Testen des Programms gespeichert.
    - `java` Java-Dateien z. B. für JUnit-Tests
    - `resources` Bilder, Icons, ...
- Bereich `target`
  Hier werden alle erzeugten Dateien (z. B. class-Dateien) gespeichert.
  Dieses Verzeichnis wird **nicht** in einem Repository gespeichert!
- ...

```
60          <!-- maven compiler -->
61          <plugin>
62             <groupId>org.apache.maven.plugins</groupId>
63             <artifactId>maven-compiler-plugin</artifactId>
64             <version>${maven.compiler.plugin.version}</version>
65             <configuration>
66                <source>${maven.compiler.source}</source>
67                <target>${maven.compiler.target}</target>
68                <encoding>${project.build.sourceEncoding}</encoding>
69                <!--
70                <testExcludes>
71                   <testExclude>**/*Test.java</testExclude>
72                </testExcludes>
73                -->
74             </configuration>
75          </plugin>
```

Das Plugin `maven-compiler-plugin` muss nur eingebunden werden, wenn explizit an den Einstellungen etwas geändert werden soll. Hier konkret, dass die Java-Version 21 und das Encoding UTF-8 verwendet werden soll. Sollen Dateien nicht übersetzt werden, so kann man `testExcludes` verwenden, welches hier auskommentiert ist.

```
78              mvn exec:java
79          -->
80          <plugin>
81             <groupId>org.codehaus.mojo</groupId>
82             <artifactId>exec-maven-plugin</artifactId>
83             <version>3.1.1</version>
84             <configuration>
85                <mainClass>${main.class}</mainClass>
86             </configuration>
87          </plugin>
88
89       </plugins>
90    </build>
91
92 </project>
```

Möchte man eine Klasse direkt über Maven starten, so hilft das Plugin `exec-maven-plugin`.

3 Projekt erstellen

1. Wechseln Sie in das Verzeichnis, in dem bei Ihnen die Projekte gespeichert werden sollen.
2. Entpacken Sie die Datei `maven_01.zip` in das Verzeichnis.
3. Wechseln Sie in dieses Verzeichnis `<projektdir>/mav01` in der Eingabeaufforderung / Konsole.
4. Erstellen Sie die notwendigen Verzeichnisse mit nachfolgendem Aufruf:

```
mgn@mars:/tmp/mav01$ mvn antrun:run@createFolder
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< de.gc.mav:TabellenTest >------------------------
[INFO] Building TabellenTest 0.0.1
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- antrun:3.0.0:run (createFolder) @ TabellenTest ---
[INFO] Executing tasks
[INFO]      [mkdir] Created dir: /tmp/mav01/src/main/resources
[INFO]      [mkdir] Created dir: /tmp/mav01/src/test/java
[INFO]      [mkdir] Created dir: /tmp/mav01/src/test/resources
[INFO] Executed tasks
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.583 s
[INFO] Finished at: 2024-04-23T08:34:17+02:00
[INFO] ------------------------------------------------------------------------
```

Die Verzeichnisse für den Sourcecode und die Ressourcendateien werden erstellt, falls diese noch nicht vorhanden sind.

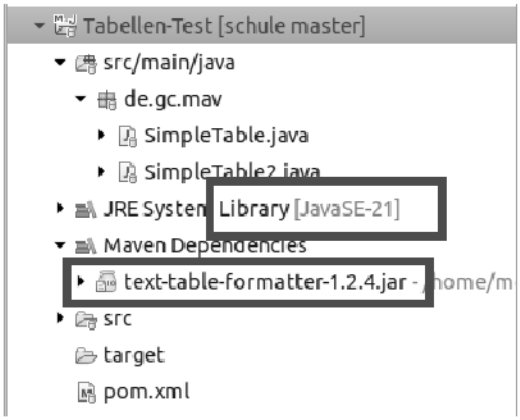
5. Nun kann der Quellcode übersetzt werden:

```
mgn@mars:/tmp/mav01$ mvn
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< de.gc.mav:TabellenTest >------------------------
[INFO] Building TabellenTest 0.0.1
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ TabellenTest ---
[INFO] Deleting /tmp/mav01/target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ TabellenTest ---
[INFO] Copying 0 resource from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.12.1:compile (default-compile) @ TabellenTest ---
[INFO] Recompiling the module because of changed source code.
[INFO] Compiling 2 source files with javac [debug target 21] to target/classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.946 s
[INFO] Finished at: 2024-04-23T08:35:58+02:00
[INFO] ------------------------------------------------------------------------
```

6. Programm starten:

```
mgn@mars:/tmp/mav01$ mvn exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< de.gc.mav:TabellenTest >------------------------
[INFO] Building TabellenTest 0.0.1
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.1.1:java (default-cli) @ TabellenTest ---
+------+-------+-----+
|Name  |Vorname|Alter|
+------+-------+-----+
|Müller|Hans   |25   |
|Meier |Uschi  |21   |
+------+-------+-----+
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.390 s
[INFO] Finished at: 2024-04-23T08:46:22+02:00
[INFO] ------------------------------------------------------------------------
```

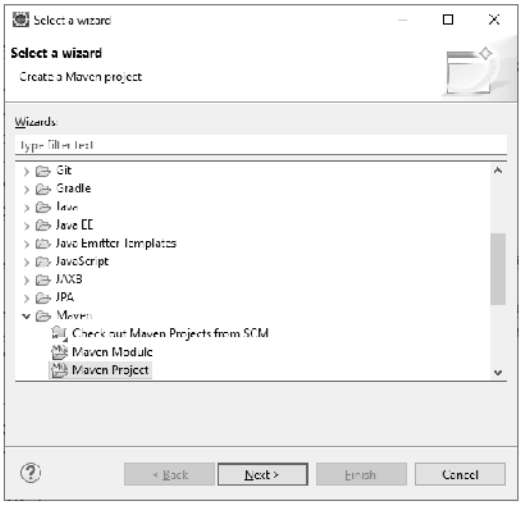

Damit die Einstellungen auch in Eclipse umgesetzt werden, wählt man mit einem Rechtsklick auf die pom-Datei im Kontextmenü Maven – Update Project... aus.



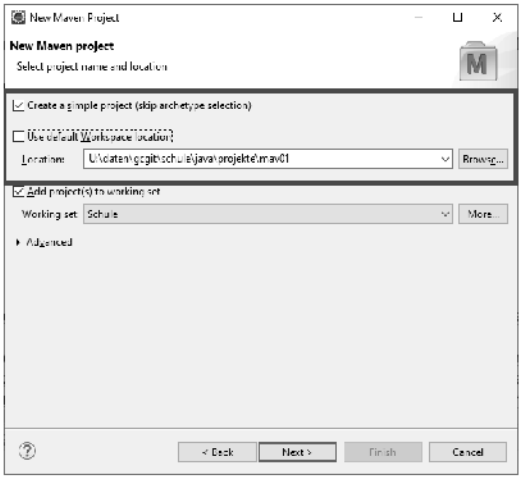
Nun ist im Projekt-Explorer die Java-Version 21 zu erkennen. Zusätzlich wird die eingebundene Bibliothek angezeigt, die standardmäßig im Home-Verzeichnis gespeichert wird.

Eclipse-Maven-Projekt erstellen

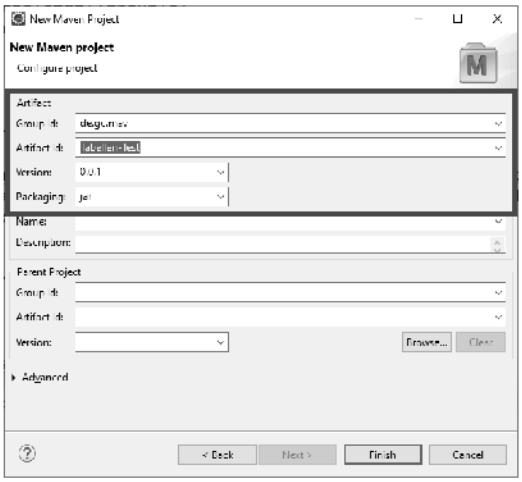
Mit nachfolgenden Schritten wird ein Maven-Projekt erstellt.



Über File – New – Other... – Maven – Maven Project wird das Projekt erstellt.



Um ein einfaches Projekt zu erstellen ist es wichtig, den Haken bei „Create a simple project...“ zu setzen, das Verzeichnis entsprechend auszuwählen und evtl. das Projekt einem „working set“ zuzuordnen.



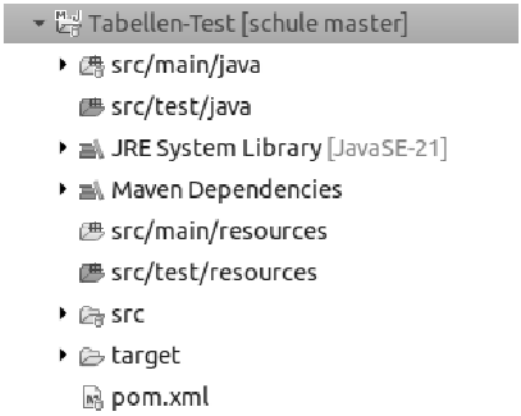
Für das Projekt müssen mindestens drei Angaben gemacht werden.

Group Id definiert den Namensraum, im Normalfall den Namespace, also der Hauptpaketname, für das Projekt.

Artifact Id legt den Namen des Projektes fest.

Version definiert die Versionsnummer für das Projekt.

Alle weiteren Einstellungen werden direkt in der Konfigurationsdatei gemacht, meist durch „copy & paste“.



Im Projekt-Explorer sieht man nun das erstellte Projekt. Dabei werden für die Hauptdateien und deren Ressourcen (Bilder, Icons, ...) ein eigener Verzeichnisbaum und für die Testfälle ein anderer definiert.

Aufgabe AUF-01-1

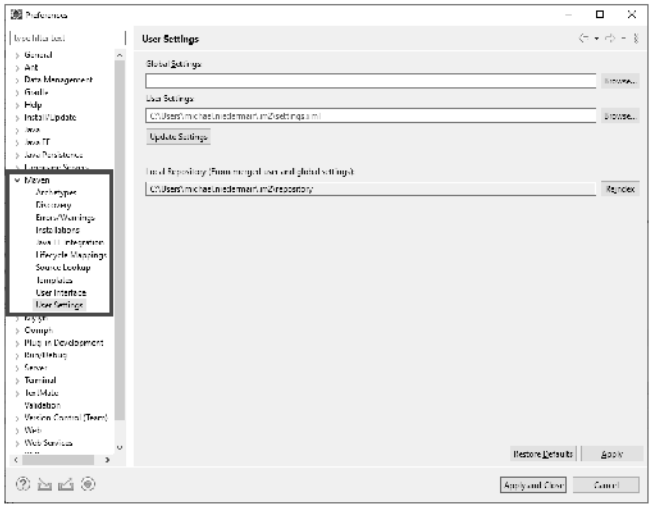
Erstellen Sie nun das Maven-Projekt, wie eben gezeigt, in Eclipse.

Notiz:

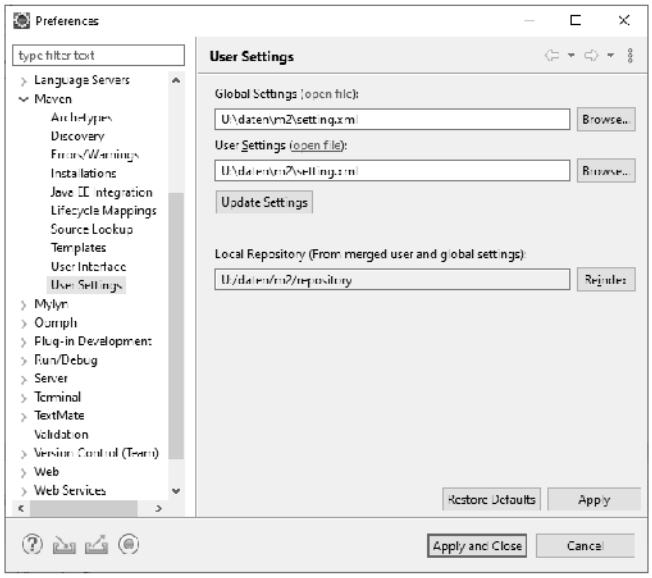
https://it.gentiana-clusii.de/ CC BY-NC-ND

Weitere Einstellungen in Maven

Maven bietet noch eine ganze weitere Reihe von Einstellungsmöglichkeiten. Diese werden in den Einstellungen unter „Preferences – Maven“ festgelegt.



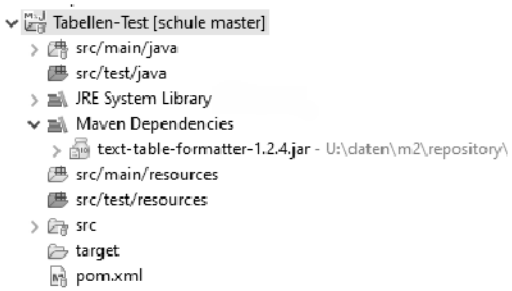
Arbeitet man mit Eclipse und einem USB-Stick, so ist es nicht immer sinnvoll, dass das lokale Repository auf dem Laufwerk `C:` gespeichert wird.



Die Einstellungen werden hier in der Datei `settings.xml` vorgenommen. Der Eintrag für ein lokales Repository findet sich unter `localRepository`. Achten Sie darauf, dass der Eintrag mit ’/’ erstellt wird, da der Backslash sonst als Escape-Sequence interpretiert wird.

`settings.xml`:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.
   apache.org/xsd/settings-1.0.0.xsd">
   <localRepository>U:/daten/m2/repository</localRepository>
   <interactiveMode />
   <offline />
   <pluginGroups />
   <servers />
   <mirrors />
   <proxies />
   <profiles />
   <activeProfiles />
</settings>
```



Das geänderte lokale Repository.

Tabellen formatiert ausgeben

Mit nachfolgendem Beispiel wird eine formatierte Tabelle erzeugt und dann ausgegeben.

```
package de.gc.mav;

import org.nocrala.tools.texttablefmt.Table;

public class SimpleTable {

   public static void main(String[] args) {

      final Table t = new Table(3);

      // header
      t.addCell("Name");
      t.addCell("Vorname");
      t.addCell("Alter");

      t.addCell("Müller");
      t.addCell("Hans");
      t.addCell("25");

      t.addCell("Meier");
      t.addCell("Uschi");
      t.addCell("21");

      System.out.println(t.render());

   }

}
```

Die Ausgabe sieht dann wie folgt aus:

```
+------+-------+-----+
|Name  |Vorname|Alter|
+------+-------+-----+
|Müller|Hans   |25   |
|Meier |Uschi  |21   |
+------+-------+-----+
```

Aufgaben AUF-01-2

1. Probieren Sie den obigen Code aus, so dass die Bibliothek auch aufgerufen wird.
2. Über den Konstruktor von `Table` lässt sich auch das Aussehen mit der Klasse `BorderStyle` festlegen. Einzelne Zellen werden mit der Klasse `CellStyle` formatiert.
   Erstellen Sie ein kleines Beispiel, welches folgende Ausgabe erzeugt. Beim „Pumukel“ wird beim Alter `null` übergeben, was aber so nicht angezeigt werden soll, dazu dient der Parameter `NullStyle.EMPTY_STRING`.

Name	Vorname	Alter
Müller	Hans	25
Meier	Uschi	21
Kobold	Pumukel	

Notiz:

4 Maven und JUnit

Mit Hilfe von Maven lassen sich auch sehr gut JUnit-Tests ausführen.

4.1 Konfiguration

Für normale JUnit-Tests werden die entsprechenden Abhängigkeiten eingebunden. Als Plugin wird zusätzlich das Maven `maven-surefire-plugin` eingebunden.

```
...
<properties>
      ...
      <junit-jupiter.version>5.10.3</junit-jupiter.version>

      <maven.compiler.plugin.version>3.12.1</maven.compiler.plugin.version>
      <maven.surefire.plugin.version>3.3.0</maven.surefire.plugin.version>

</properties>

<dependencies>
      ...

      <!--  JUNIT    -->
      <dependency>
         <groupId>org.junit.jupiter</groupId>
         <artifactId>junit-jupiter</artifactId>
         <scope>test</scope>
         <version>${junit-jupiter.version}</version>
      </dependency>

</dependencies>

<build>
      <defaultGoal>clean compile install</defaultGoal>

      <plugins>
         ...

         <!-- Testausgabe: TXT, XML -->
         <!-- mvn test -->
         <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>${maven.surefire.plugin.version}</version>
         </plugin>

      </plugins>
</build>
```

Notiz:

4.2 Test ausführen

Über den Aufruf in der Konsole werden alle Tests ausgeführt. Gibt man nichts anderes an, so werden die Ergebnisse im Verzeichnis `target/surefire-reports` gespeichert.

```
> mvn test

[INFO] Scanning for projects...
...
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running de.gc.hv.db.DbConTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.409 s
 -- in de.gc.hv.db.DbConTest
[INFO] Running de.gc.hv.db.DbInsertTest
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.214 s
 -- in de.gc.hv.db.DbInsertTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 13, Failures: 0, Errors: 0, Skipped: 0
...
```

Für die beiden Beispieltests wurden insgesamt vier Dateien erzeugt. Pro Test wird eine TXT- und eine XML-Datei erzeugt. Die XML-Datei kann für eine weitere Auswertung verwendet werden.

```
-------------------------------------------------------------------------------
Test set: de.gc.hv.db.DbConTest
-------------------------------------------------------------------------------
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.409 s -- in de.gc.hv.db.DbConTest
```

Die Textausgabe ist für einfache Sachen „ganz nett“, aber bei vielen Tests verliert man hier schnell die Übersicht.

Notiz:

4.3 HTML-Ausgabe

Für die HTML-Ausgabe von Tests fügt man einen neuen Bereich mit den Namen „reporting“ in der Konfigurationsdatei hinzu. Hier wird das Plugin `maven-surefire-report-plugin` verwendet.

```
...
<reporting>
   <plugins>

      <!-- mvn surefire-report:report -->
      <!-- HTML-Ausgabe -->
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-surefire-report-plugin</artifactId>
         <version>${maven.surefire.plugin.version}</version>
      </plugin>

   </plugins>
</reporting>
```

Über den Aufruf `mvn surefire-report:report` in der Konsole werden hier alle Tests, wie vorher auch, durchgeführt. Als Ergebnis erhält man im Verzeichnis `target/site` eine entsprechende HTML-Seite mit CSS und Bildern.

Surefire Bericht

Zusammenfassung

[Zusammenfassung] [Pakete] [Testfälle]

Tests	Fehler	Fehlschläge	Ausgelassen	Erfolgsrate	Zeit
13	0	0	0	100 %	2,751 s

Hinweis: Fehlschläge werden erwartet und durch Behauptungen überprüft während Fehler unerwartet sind.

Pakete

[Zusammenfassung] [Pakete] [Testfälle]

Paket	Tests	Fehler	Fehlschläge	Ausgelassen	Erfolgsrate	Zeit
de.gc.hv.db	13	0	0	0	100 %	2,751 s

Hinweis: Die Paketstatistiken werden nicht rekursiv berechnet, es werden lediglich die Ergebnisse aller enthaltenen Tests aufsummiert.

de.gc.hv.db

-	Klasse	Tests	Fehler	Fehlschläge	Ausgelassen	Erfolgsrate	Zeit
⚠	DbConTest	4	0	0	0	100 %	0,402 s
⚠	DbInsertTest	9	0	0	0	100 %	2,349 s

Testfälle

[Zusammenfassung] [Pakete] [Testfälle]

DbConTest

⚠	t_01_connection	0,250 s
⚠	t_02_dropAllTables	0,032 s
⚠	t_03_createAllTables	0,055 s
⚠	t_04_showTables	0,022 s

4.4 Testabdeckung – code coverage

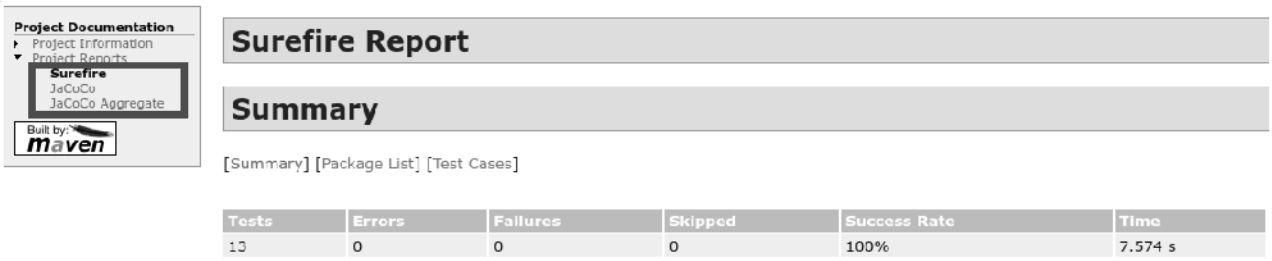
Mit Maven lässt sich auch die Testabdeckung ermitteln. Dazu wird das Plugin `jacoco-maven-plugin` verwendet.

```
...
   <properties>
      ...
      <jacoco.version>0.8.12</jacoco.version>
   </properties>

   <profiles>
      <profile>
         <id>test</id>
         <properties>
            <env>test</env>
            <gebEnv>test</gebEnv>
            <jacoco.skip>false</jacoco.skip>
            <maven.test.skip>false</maven.test.skip>
            <skip.unit.tests>false</skip.unit.tests>
         </properties>
      </profile>
   </profiles>

   <build>
      ...
      <plugins>
      ...
         <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>${jacoco.version}</version>
            <executions>
               <execution>
                  <goals>
                     <goal>prepare-agent</goal>
                  </goals>
               </execution>
               <execution>
                  <id>report</id>
                  <phase>prepare-package</phase>
                  <goals>
                     <goal>report</goal>
                  </goals>
               </execution>
            </executions>
         </plugin>

      </plugins>
   </build>

   <reporting>
      <plugins>

         <!-- mvn surefire-report:report -->
         <!-- HTML-Ausgabe -->
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-report-plugin</artifactId>
            <version>${maven.surefire.plugin.version}</version>
         </plugin>

         <!-- Coverage -->
         <!-- mvn clean install site -P test -->
         <plugin>
```

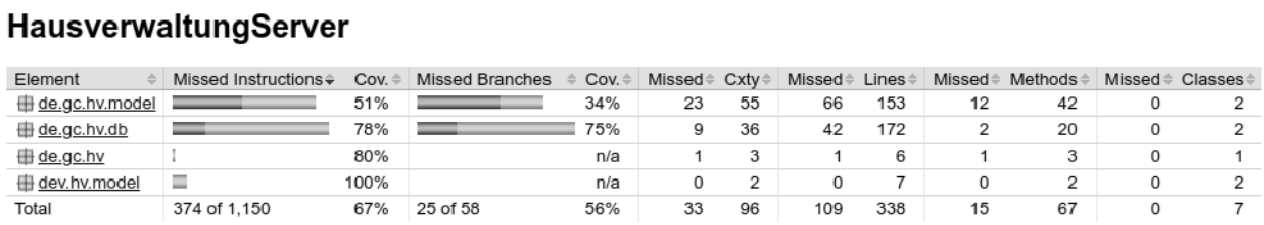

```
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>${jacoco.version}</version>
        </plugin>

    </plugins>
  </reporting>
```

Mit dem Aufruf `mvn clean install site -P test` wird u.a. zuerst das Verzeichnis `target` gelöscht und dann alle Dateien erneut übersetzt, um sicher zu gehen, dass die Testabdeckung immer mit dem „selben Anfangszustand“ beginnt. Im Anschluss wird die Testabdeckung mit dem Profile `test` gestartet. Das Ergebnis wird dann in der selben HTML-Datei zu den normalen Tests hinzugefügt.

Mit **`mvn clean install surefire-report:report site -P test`** wird die HTML-Ausgabe für die Tests und die Testabdeckung in einem „Rutsch“ erstellt.



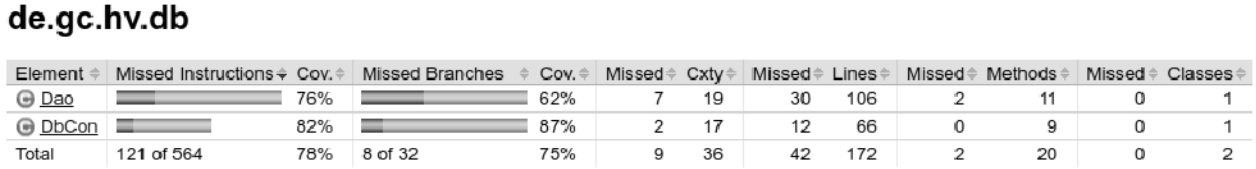
Klickt man auf „JaCoCo“, so wird das Ergebnis der Testabdeckung angezeigt.



HausverwaltungServer

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
de.gc.hv.model		51%		34%	23	55	66	153	12	42	0	2
de.gc.hv.db		78%		75%	9	36	42	172	2	20	0	2
de.gc.hv		80%		n/a	1	3	1	6	1	3	0	1
dev.hv.model		100%		n/a	0	2	0	7	0	2	0	2
Total	374 of 1,150	67%	25 of 58	56%	33	96	109	338	15	67	0	7

Klickt man auf ein entsprechendes Package, so erhält man detaillierte Angaben.



de.gc.hv.db

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Dao		76%		62%	7	19	30	106	2	11	0	1
DbCon		82%		87%	2	17	12	66	0	9	0	1
Total	121 of 564	78%	8 of 32	75%	9	36	42	172	2	20	0	2

Notiz:

5 Programme / Klassen über Maven ausführen / starten

Oft ist es erwünscht, direkt über Maven das entwickelte Programm zu starten.

```
...
   <properties>
      ...
      <main.class>rest.Server.Main</main.class>
   </properties>

   <build>
      ...
      <plugins>
         ...
         <!-- execute
            mvn exec:java@run-server
            see https://www.mojohaus.org/exec-maven-plugin/java-mojo.html
         -->
         <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>exec-maven-plugin</artifactId>
            <version>3.3.0</version>
            <executions>
               <execution>
                  <id>run-server</id>
                  <goals>
                     <goal>java</goal>
                  </goals>
                  <configuration>
                     <mainClass>${main.class}</mainClass>
                  </configuration>
               </execution>
            </executions>
         </plugin>
      ...
      </plugins>
   </build>
```

Hier wird über den Aufruf `mvn exec:java@run-server` die Klasse `rest.Server.Main` direkt über Maven gestartet (diese muss natürlich eine `main()`-Methode haben).

Notiz:

6 Ein Archiv erzeugen

Das Assembly-Plug-In für Maven ermöglicht es, Projektdateien in einem einzigen verteilbaren Archiv zu kombinieren, das auch Abhängigkeiten, Module, ... und andere Dateien enthalten kann.

```
...
   <build>
      ...
      <plugins>

         <!--
            run without tests
            mvn assembly:single -DskipTests
            see https://maven.apache.org/plugins/maven-assembly-plugin/
         -->
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-assembly-plugin</artifactId>
            <version>3.3.0</version>

            <executions>
               <execution>
                  <id>make-schueler</id>
                  <phase>package</phase>
                  <goals>
                     <goal>single</goal>
                  </goals>
                  <configuration>
                     <descriptors>
                        <descriptor>src/assembly/schueler_java.xml</descriptor>
                        >
                     </descriptors>
                  </configuration>
               </execution>
            </executions>
         </plugin>

      ...
      </plugins>
   </build>
```

In der Datei `schueler_java.xml` wird dann genau definiert, welche Dateien etc. eingebunden werden.

Damit nicht jedes mal alle Tests ausgeführt werden, können diese über `-DskipTests` für diesen Aufruf deaktiviert werden.

```
<assembly xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/
assembly/1.1.3"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-
          plugin/assembly/1.1.3 http://maven.apache.org/xsd/assembly-1.1.3.xsd
          ">
    <id>zip-schueler</id>
    <formats>
        <format>zip</format>
    </formats>
    <fileSets>
        <fileSet>
            <directory>${project.basedir}</directory>
            <includes>
                <include>src/main/java/model/*.java</include>
                <include>src/main/java/vorlage/**</include>
                <include>src/test/resources/*.json</include>
            </includes>
```


```
            <outputDirectory>/</outputDirectory>
        </fileSet>
    </fileSets>
</assembly>
```

Notiz: