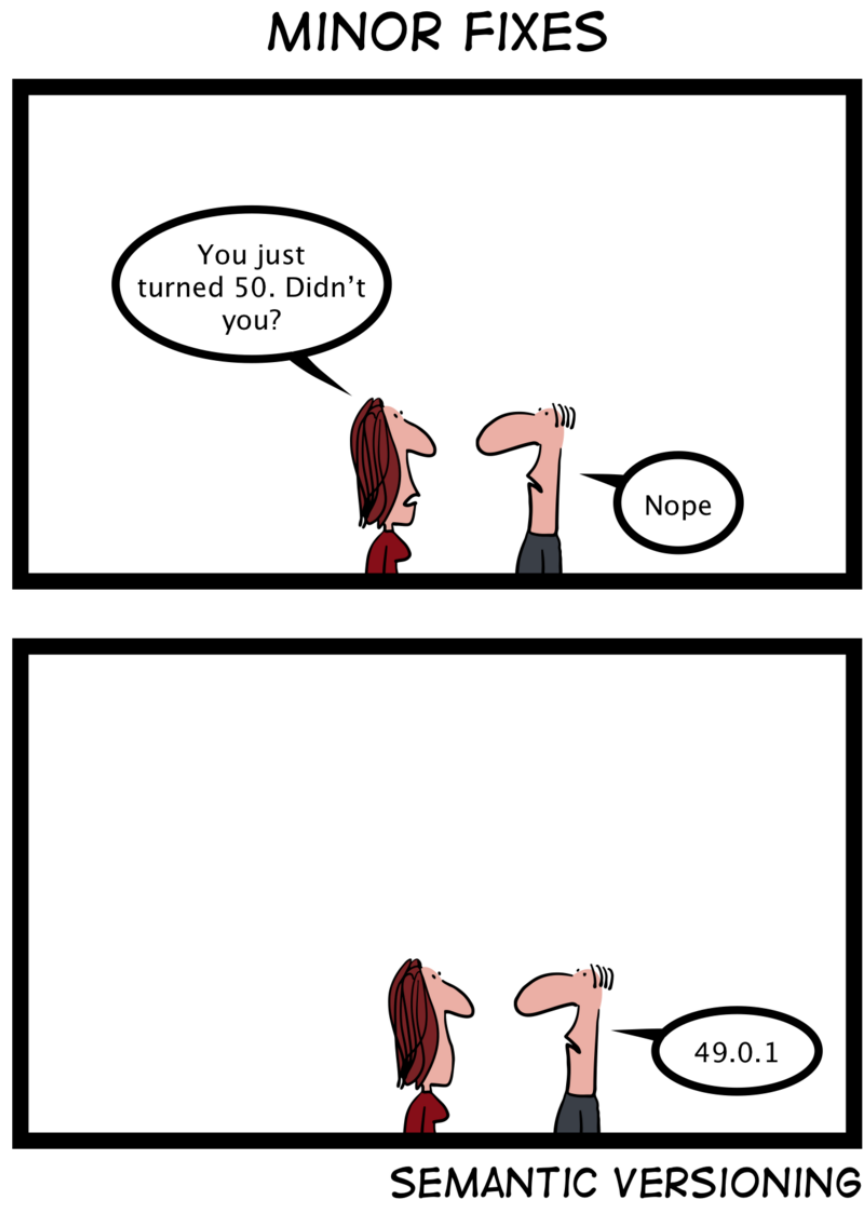
BuildTools

Vorlesungsinhalt



- Überblick
- Maven
- Maven Softwarepaket
- Maven Lifecycle
- Maven in IntelliJ
- Exkurs: Maven in Eclipse
- Aufgabe

Überblick

Build Tools

Software

- entsteht aus Quellcode (Kompilierung)
- besteht oft aus mehreren von einander abhängigen Modulen
- bindet Ressourcen ein (z.B. Bilder)
- kann unterschiedliche externe Bibliotheken benötigen
- kann bestimmte Schritte bei der Erstellung erfordern

Insbesondere in der Entwicklungsphase sind diese Aspekte zu beachten.

Build Tools

Idee: dies lässt sich automatisieren

- Im C-Umfeld spricht man von „makefiles“
- Im Java-Umfeld wurde dafür „Ant“ entwickelt (circa im Jahr 2000) `http://ant.apache.org`
  - Ähnelt klassischen makefiles
  - Sehr mächtig, XML als Sprache
- Auf „Ant“ folgte „Maven“, basierend auf XML und zentralen Datenbanken für externe Abhängigkeiten
- Auf „Maven“ folgte „Gradle“ `http://gradle.org`
  - Auf Basis der Konzepte von Ant und Maven entstanden
  - Groovy statt XML

Maven hat den höchsten Verbreitungsgrad → In der Vorlesung wird Maven genutzt

Ant Beispiel

```
<project>
  <target name="clean">
    <delete dir="classes" />
  </target>
  <target name="compile" depends="clean">
    <mkdir dir="classes" />
    <javac srcdir="src" destdir="classes"
        />
  </target>
  <target name="jar" depends="compile">
    <mkdir dir="jar" />
    <jar destfile="jar/HelloWorld.jar"
        basedir="classes">
      <manifest>
        <attribute name="Main-Class"
          value="antExample.HelloWorld" /
            >
      </manifest>
    </jar>
  </target>
</project>
```

`ant compile`

- Führe `target` compile aus, dieses führt zunächst `target` clean aus.

`ant jar`

- Führe `target` jar aus, dieses führt zunächst `target` compile aus.

Ant Beispiel

Anhand des Ant Beispiels wird gut ersichtlich:

- Es gibt einzelne Aufgaben, die systematisch erledigt werden sollen
- Diese Aufgaben sind nicht unabhängig voneinander
- Alle als vorausgesetzte Aufgaben müssen vor der Erledigung einer abhängigen Aufgabe ausgeführt werden
- Einzelne Aufgaben können direkt angesteuert werden

Gradle Beispiel

```
apply plugin: 'java'

repositories {
    mavenCentral()
}

jar {
    baseName = 'gradleExample'
    version = '1.0.0-SNAPSHOT'
}

dependencies {
    compile 'junit:junit:4.12'
}
```

`gradle classes`

- Baut die Software und erstellt eine JAR-Datei
- Die Targets von Ant können hier als Aufgaben (tasks) angegeben werden

Gradle Beispiel

Anhand des Gradle Beispiels wird gut ersichtlich:

- Es gibt die Möglichkeit externe Repositories anzugeben (hier MavenCentral) - diese werden für die Auflösung der Abhängigkeiten verwendet
- Beschreibungen in Gradle sind sehr schlank
- Fast die gesamte Funktionalität wird über Plug-Ins bereitgestellt
- Entwickler müssen eine neue Sprache lernen

Maven

Maven

- Build-Management-Tool der Apache Software Foundation
- „Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.“[1]
- https://maven.apache.org/

[1] https://maven.apache.org/, Januar 2019

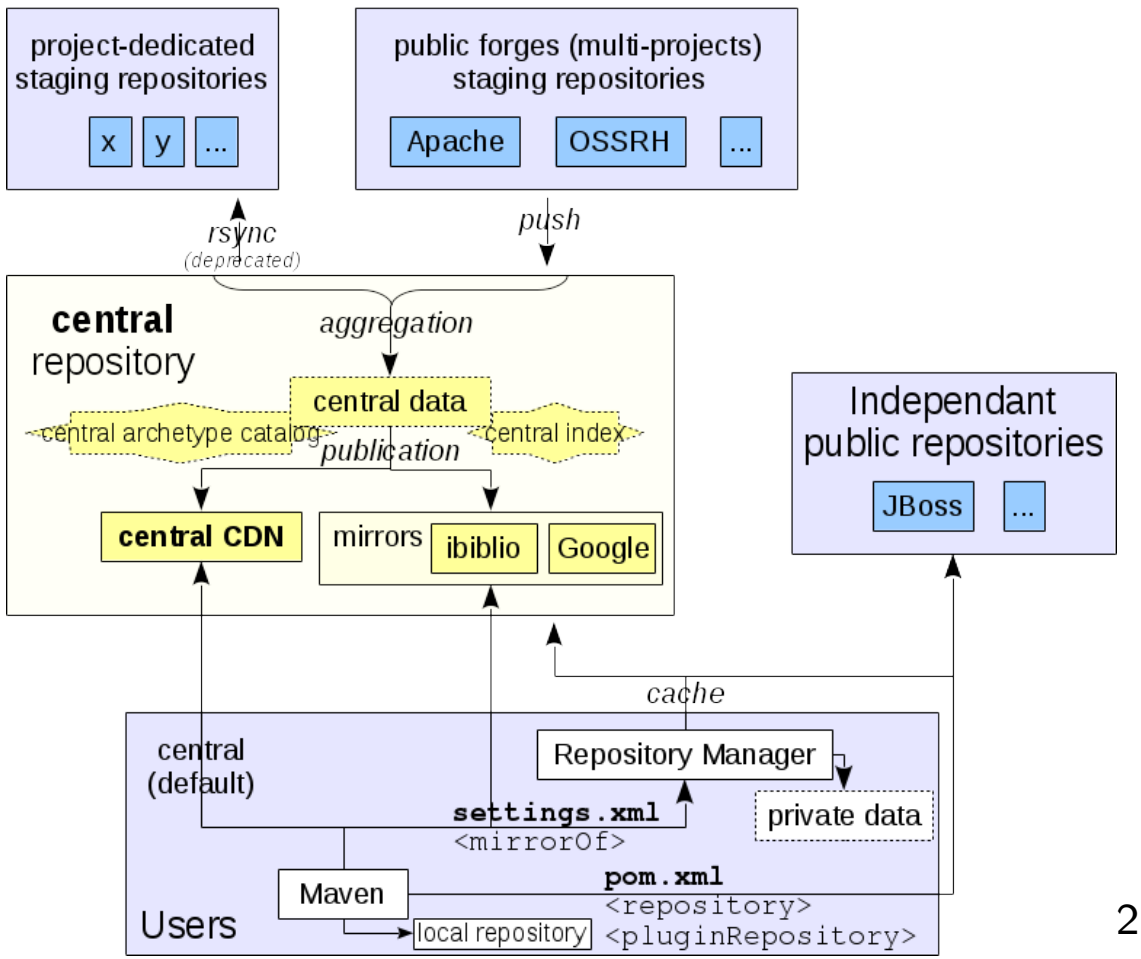
Maven - Verzeichnisstruktur

Maven sieht eine Standard Verzeichnisstruktur vor.

```
+---src
|   +---main
|   |   +---java
|   |   |
|   |   \---resources
|   |
|   \---test
|       +---java
|       |
|       \---resources
|
\---pom.xml
```

Maven - Repositories

In Repositories werden wiederverwendbare Artefakte in beliebig vielen Versionen vorgehalten.



[2]

[2] Grafik unter https://maven.apache.org/repository/

Maven - POM Dateien

Project Object Model

- Die `pom.xml` Datei eines Projektes beinhaltet alle relevanten Informationen
- Ein Maven-Projekt muss keinen Quellcode enthalten: eine `pom.xml` Datei genügt
- Diese POM-Datei beinhaltet[3]
  - Alle Abhängigkeiten
  - Identifikatoren
  - Zugehörigkeit zu anderen POM Dateien
  - Die enthaltenen Module
  - Eigenschaften
  - Build Einstellungen
  - Weitere Projektinformationen
  - Umgebungseinstellungen

[3] siehe auch https://maven.apache.org/pom.html

Maven - POM Dateien

XML/Modell Einstellungen

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
```

- Die Attribute von `project` verweisen auf die POM Dateien zugrundeliegenden Definitionen (XML Schema) und sind entsprechend der verwendeten POM zu setzen
- Die `modelVersion` ist fest auf 4.0.0 zu setzen.

Maven - POM Dateien

Grundlegende Angaben, Identifikation der Software

```
<!-- The Basics -->
<groupId>...</groupId>
<artifactId>...</artifactId>
<version>...</version>
<packaging>...</packaging>
...
```

- `groupId`: (**verpflichtend**) eindeutig zu wählen (innerhalb einer Organisation / eines Projektes), z.B. `org.jamesii` - sollte mit dem Pfad des Projektes auf dem Datenträger korrespondieren
- `artifactId`: (**verpflichtend**) der „eigentliche“ Name des Projekts
- `version`: (**verpflichtend**) zur Unterscheidung verschiedener Versionen von `groupId:artifactId`
- `packaging`: definiert, wie das Projekt gepackt werden soll (z.B. jar [default], war)
- `classifier`: weitere Unterscheidung von Produkten (z.B. gebaut für unterschiedliche Java Versionen)

Vollständig: `groupId:artifactId:packaging:classifier:version`

Maven - POM Dateien

Grundlegende Angaben, Beziehungen

```
<!-- The Basics -->
...
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</
        artifactId>
    <version>4.0</version>
    <type>jar</type>
    <scope>test</scope>
    <optional>true</optional>
  </dependency>
  ...
</dependencies>
...
```

Die Syntax der Versionsangaben

- 1.0: „Weiche“ Festlegung auf 1.0 (Empfehlungscharakter)
- [1.0]: „Harte“ Festlegung auf 1.0
- (, 1.0]: $x \leq 1.0$
- [1.2, 1.3]: $1.2 \leq x \leq 1.3$
- [1.0, 2.0): $1.0 \leq x < 2.0$
- [1.5,): $x \geq 1.5$
- (, 1.0], [1.2,): $x \leq 1.0$ or $x \geq 1.2$; mehrere Mengen werden durch Kommata separiert
- (, 1.1), (1.1,): dies schließt 1.1 aus (z.B., wenn eine Inkompatibilität zu dieser bekannt ist)

- `dependencies`: Beinhaltet beliebig viele einzelne Abhängigkeiten
- `dependency`: Beinhaltet die identifizierenden Angaben `groupId`, `artifactId`, `version` für das Paket, zu dem die Abhängigkeit besteht, sowie Informationen über den Zeitpunkt der Abhängigkeit
- `type`: jar [default], entspricht dem `packaging` Typ der `dependency`
- `scope`: beschränkt die Gültigkeit (Zeit) der `dependency` (`compile`, `provided`, `runtime`, `test`, `system`))
- `optional`: Zeigt für Verwender dieses Projekts an, dass diese Abhängigkeit nicht aufgelöst werden muss

Maven - POM Dateien

Grundlegende Angaben

```
<!-- The Basics -->
...
<parent>...</parent>
<dependencyManagement>...</dependencyManagement>
<modules>...</modules>
<properties>...</properties>
```

- `parent`: Unterstützung für Projektvererbung, fast alle Elemente der parent-pom werden ererbt (abgesehen von artifactId, name und prerequisites), Interessant für Projekte, die aus mehreren Teilprojekten bestehen
- `dependencyManagement`: Verwaltung von Abhängigkeiten über die Vererbung hinweg
- `modules`: Aggregation aus mehreren Modulen
- `properties`: Konstantendefinition zur Verwendung in der POM

Maven - POM Dateien

Einstellung für das Bauen der Software 1/4

```
<!-- Build Settings -->
<build>
  <defaultGoal>install</defaultGoal>
  <directory>${basedir}/target</directory>
  <finalName>${artifactId}-${version}</finalName>
  <filters>
    <filter>filters/filter1.properties</filter>
  </filters>
</build>
<reporting>...</reporting>
```

- `build`: Beinhaltet die „build“ Einstellungen
- `install`: Default der auszuführenden Phase
- `directory`: Zielverzeichnis, default ist `${basedir}/target`
- `finalName`: Name der zu erstellenden Datei (ohne Endung), default ist `${artifactId}-${version}`
- `filter`: Definiert *.properties Dateien, die Variablen in den Projektdateien durch konkrete Werte ersetzen

Maven - POM Dateien

Einstellung für das Bauen der Software 2/4

```
<!-- Build Settings -->
<build>
    ...
    <resources>
        <resource>
            <targetPath>META-INF/plexus</targetPath>
            <filtering>false</filtering>
            <directory>${basedir}/src/main/plexus</directory>
            <includes>
                <include>configuration.xml</include>
            </includes>
            <excludes>
                <exclude>**/*.properties</exclude>
            </excludes>
        </resource>
    </resources>
    <testResources>
        ...
    </testResources>
    ...
</build>
```

- `resources`: Beinhaltet Angaben zu Ressourcen
- `testResources`: Beinhaltet Angaben zu den Ressourcen für die Testphase, diese werden nicht veröffentlicht

Maven - POM Dateien

Einstellung für das Bauen der Software 3/4

```
<!-- Build Settings -->
<build>
  ...
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>2.6</version>
      <extensions>false</extensions>
      <inherited>true</inherited>
      <configuration>
        <classifier>test</classifier>
      </configuration>
      <dependencies>...</dependencies>
      <executions>...</executions>
    </plugin>
  </plugins>
</build>
```

- `plugins`: Beinhaltet Angaben zu Maven Plug-Ins, die für das Bauen des Projektes verwendet werden sollen

Maven - POM Dateien

Einstellung für das Bauen der Software 4/4

```
<!-- Build Settings -->
<build>
  ...
  <reporting>
    <outputDirectory>${basedir}/target/site</outputDirectory
        >
    <plugins>
      <plugin>
        <artifactId>maven-project-info-reports-plugin</
            artifactId>
        <version>2.0.1</version>
        <reportSets>
          <reportSet></reportSet>
        </reportSets>
      </plugin>
    </plugins>
  </reporting>
</build>
```

- `reporting`: Beinhaltet Angaben zu Reports, die von Maven Plug-Ins erstellt werden können (z.B. wo diese abgelegt werden sollen), Beispiele sind: Übersicht der Abhängigkeiten, des Teams oder ggf. Analyseergebnisse

Maven - POM Dateien

Weitere Projektinformationen

```
<!-- More Project Information -->
<name>...</name>
<description>...</description>
<url>...</url>
<inceptionYear>...</inceptionYear>
<licenses>...</licenses>
<organization>...</organization>
<developers>...</developers>
<contributors>...</contributors>
```

Maven - POM Dateien

Umgebungseinstellungen

```
  <!-- Environment Settings -->
  <issueManagement>...</issueManagement>
  <ciManagement>...</ciManagement>
  <mailingLists>...</mailingLists>
  <scm>...</scm>
  <prerequisites>...</prerequisites>
  <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
  <distributionManagement>...</distributionManagement>
  <profiles>...</profiles>
</project>
```

Maven - POM Dateien

Zusammenfassung

- Maven und die POM-Dateien bieten vielseitige Konfigurationsmöglichkeiten
- Durch Plug-Ins kann die Funktionalität stark erweitert werden
- POM-Dateien zentralisieren Informationen
- Durch die Angaben in den POM-Dateien, organisiert Maven die Abhängigkeiten
- Die Webseite unter `https://maven.apache.org/pom.html` beinhaltet sehr viel mehr Informationen

Maven Softwarepaket

Maven

Maven ist eine eigenständige Software, die heruntergeladen und installiert werden kann

- Die Maven-Version dieser Installation sollte mit der in den IDEs verwendeten Maven-Version übereinstimmen / kompatibel sein
- Die Konfiguration der IDE Maven Integration und der eigenständigen Maven Installation müssen aufeinander abgestimmt sein
- Maven legt ein lokales (projektübergreifendes) Repository an, in dem die heruntergeladenen Artefakte vorgehalten werden: ACHTUNG - das entsprechende Verzeichnis wächst bei intensiver Nutzung ständig an.
- Wir verwenden Maven in einer aktuellen Version
- Offizielle Webseite zum Download von Maven: `https://maven.apache.org/`

Maven - Benutzung

- Maven ist eine Kommandozeilenanwendung
- Der Aufruf von Maven erfolgt über den Befehl `mvn`
- Für die meisten Projekte genügt ein Aufruf von `mvn verify` zum Bauen
- Ggf. ist ein `mvn clean verify` erforderlich: hier wird zunächst das Ausgabeverzeichnis geleert
- Informationen zum Ausführen von Maven: `https://maven.apache.org/run-maven/index.html`

Maven Lifecycle

Maven Lifecycle

- Maven beinhaltet standardmäßig einen Build Lifecycle
- Build Lifecycle ist der Prozess zum Bauen und Verteilen eines Artefaktes
- Maven unterscheidet drei Lifecycles ‘clean’, ‘default’ und ‘site’

Maven - Clean Lifecycle

Löscht die generierten Dateien vorangegangener Builds. Hat folgende Phasen:

pre-clean	Prozesse ausführen, die vor der eigentlichen Projektreinigung benötigt werden.
clean	Entfernt alle Dateien, die durch die vorherigen Builds erzeugt wurden.
post-clean	Prozesse ausführen, die nach der eigentlichen Projektreinigung benötigt werden.

Maven - Default Lifecycle 1/3

Standard Lifecycle zum Bauen eines Artefakts mit 23 Phasen:

validate	Validerung des Projektes und Prüfung, ob Informationen vorliegen
initialize	Initialisierung des Build Zustands (z.B. Verzeichnisse erstellen)
generate-sources, process-sources	Generierung von Quellcode und Vorverarbeitung von diesem (z.B. Filtern von Werten)
generate-resources, process-resources	Generierung von Ressourcen und Vorverarbeitung von diesen (z.B. Kopieren von Ressourcen)
compile, process-classes	Kompilieren vom Quellcode, Nachverarbeitung von kompilierten Dateien

Maven - Default Lifecycle 2/3

generate-test-sources, process-test-sources	Generierung von Test-Quellcode und Vorverarbeitung von diesem
generate-test-resources, process-test-resources	Generierung von Test-Ressourcen und Vorverarbeitung von diesen
test-compile, process-test-classes	Kompilieren vom Test-Quellcode, Nachverarbeitung von kompilierten Dateien
test	Ausführen von Unit Test Frameworks (z.B. jUnit Tests)
prepare-package, package	Vorverarbeitung und Verpacken des kompilierten Quellcodes in das definierte Format (z.B. JAR)

Maven - Default Lifecycle 3/3

pre-integration-test, integration-test, post-integration-test	Vorverarbeitung, Durchführung und Nachverarbeitung von Integrationstests
verify	Überprüfung, ob Package valide ist und Qualitätskriterien erfüllt
install	Ablegen des Artefaktes in das lokale Maven Repository
deploy	Ablegen des Artefaktes in das remote Maven Repository zum Teilen mit anderen Entwicklern

Maven - Site Lifecycle

Phase zum Generieren einer Dokumentation zum Projekt.

pre-site	Ausführen von Prozesses vor der Generation
site	Generieren der Dokumentation für das Projekt
post-site	Ausführen von Prozesses nach der Generation
site-deploy	Ausliefern der generierten Dokumentation

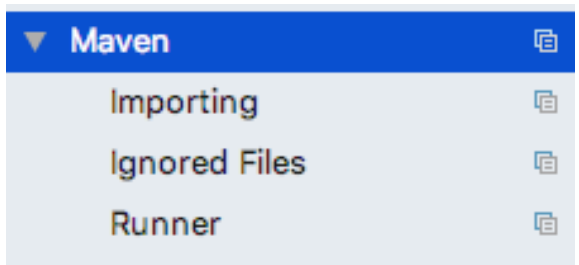
Maven in IntelliJ

Maven in IntelliJ

- IntelliJ baut die Software, wenn dies benötigt ist
- In Maven definierte Abhängigkeiten und die durch Maven Plug-Ins erreichten und in einem Projekt verwendeten Fähigkeiten müssen auch innerhalb von IntellJ ausgewertet werden. (Oder durch Verdopplung des Aufwandes in IntelliJ soweit möglich nachgebaut werden).

Maven - Konfiguration

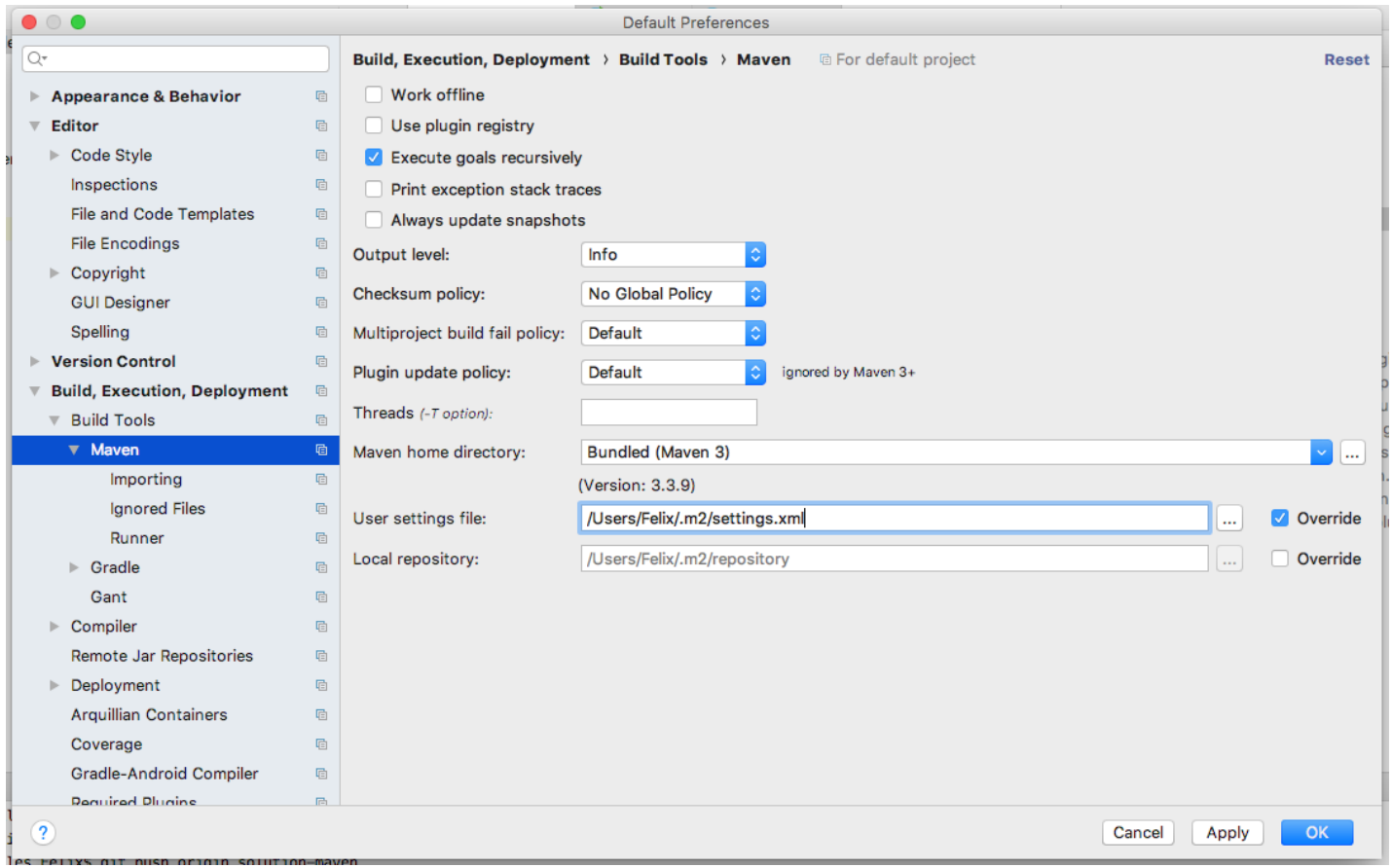
File → *Settings* → *Build*, *Execution*, *Deployment* → *Maven*



Maven - Konfiguration

Maven Benutzereinstellungen

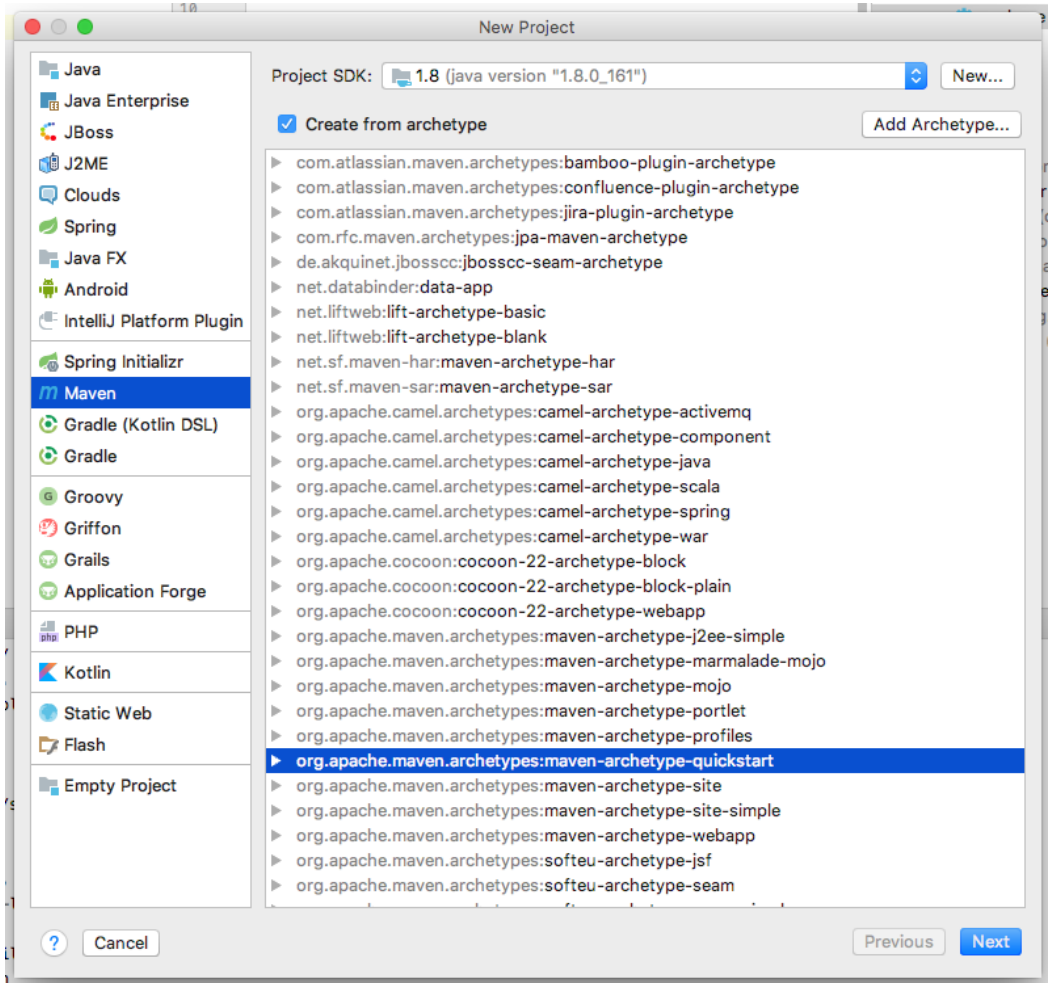
File → *Settings* → *Build*, *Execution*, *Deployment* → *Maven*



Maven - Neues Projekt

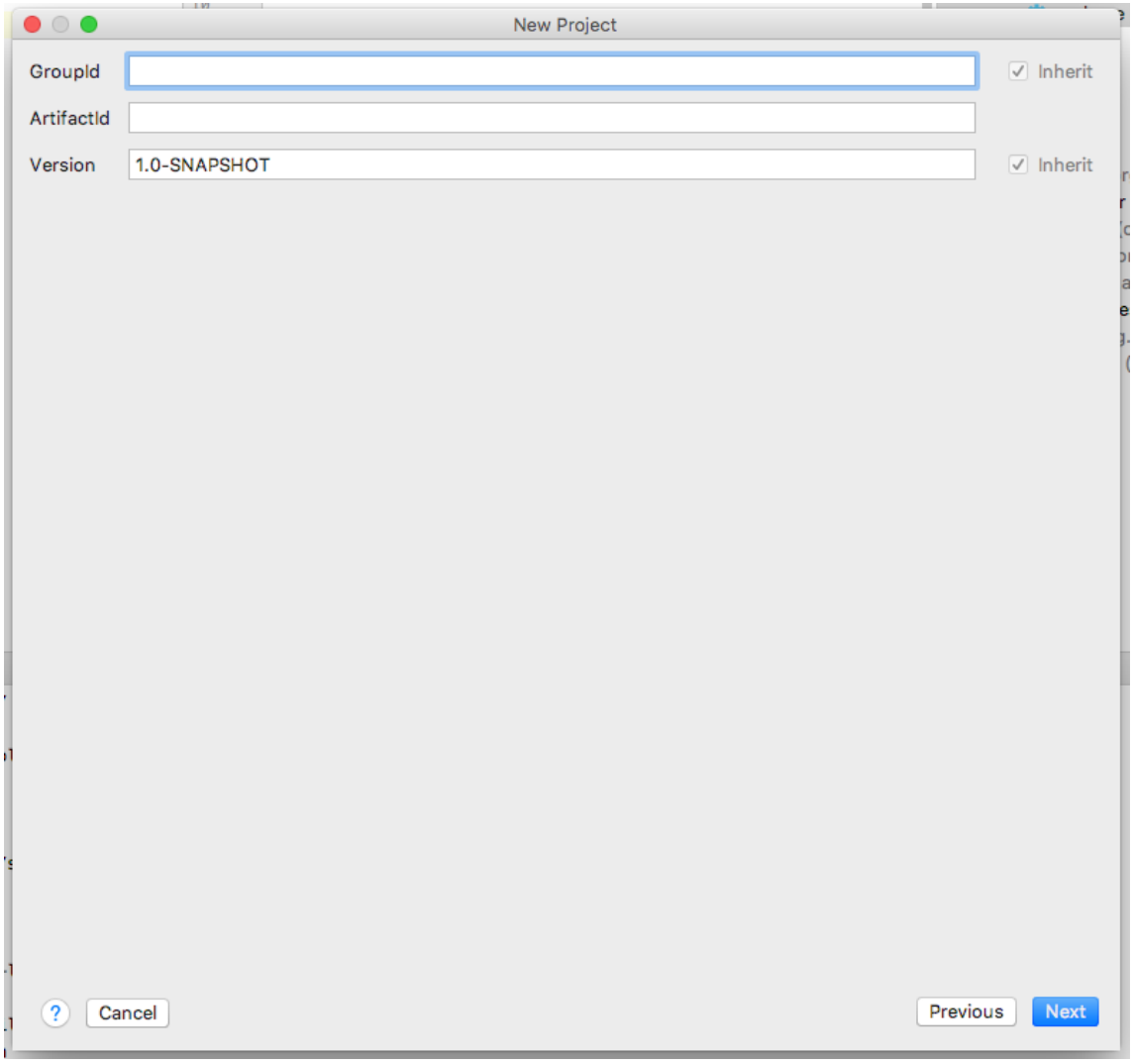
Auswahl eines Maven Projekts

File → *New* → *Project...* → *Maven*



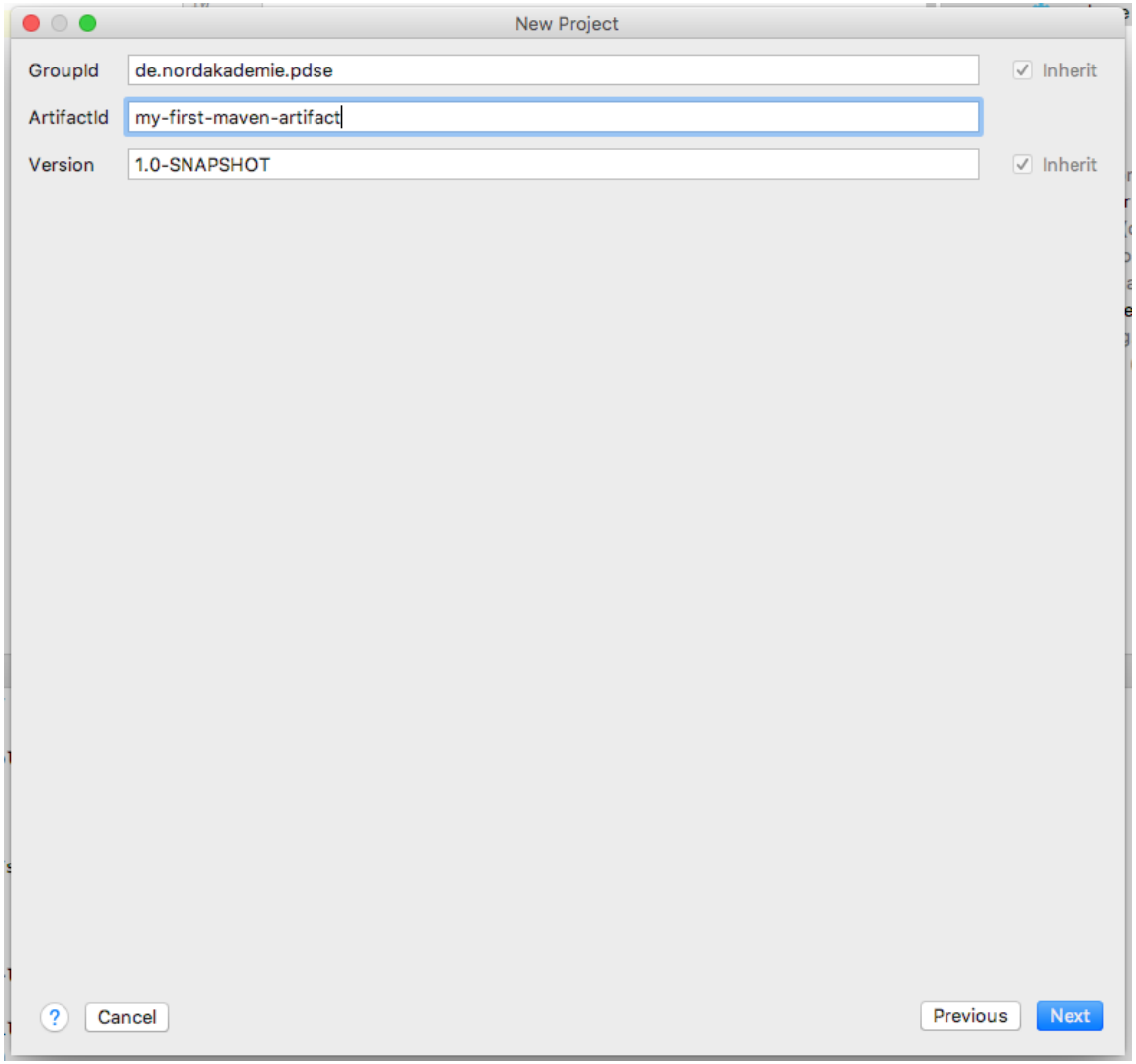
Maven - Neues Projekt

Identifikation des Projekts



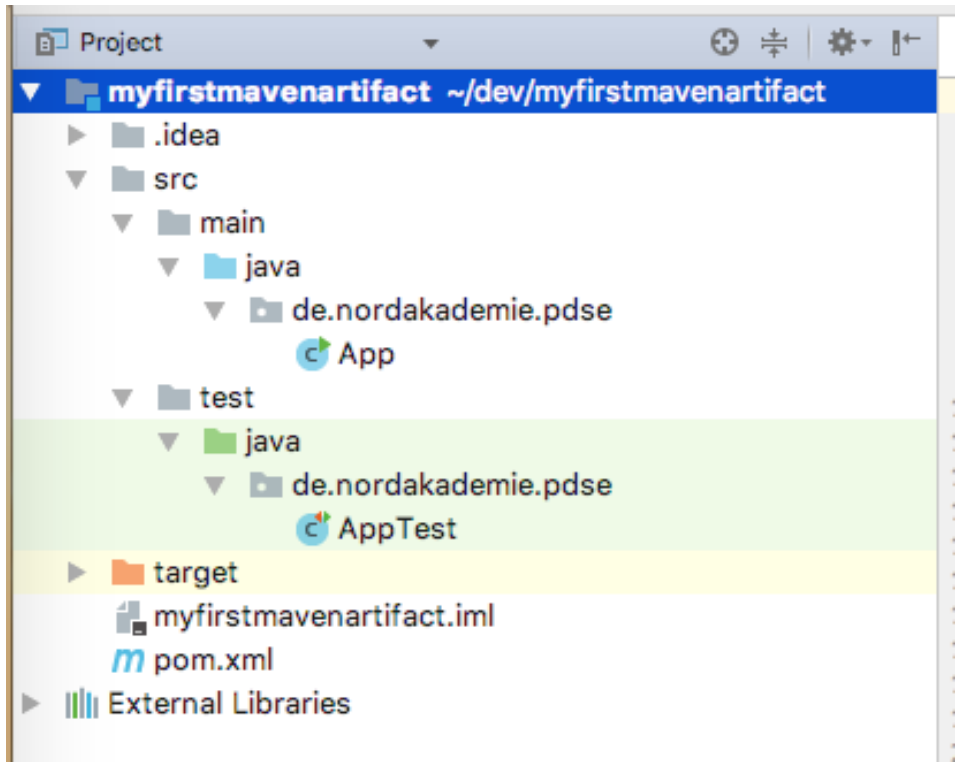
Maven - Neues Projekt

Identifikation des Projekts, Bsp



Maven - Neues Projekt

Erstellt, im Package Explorer offen



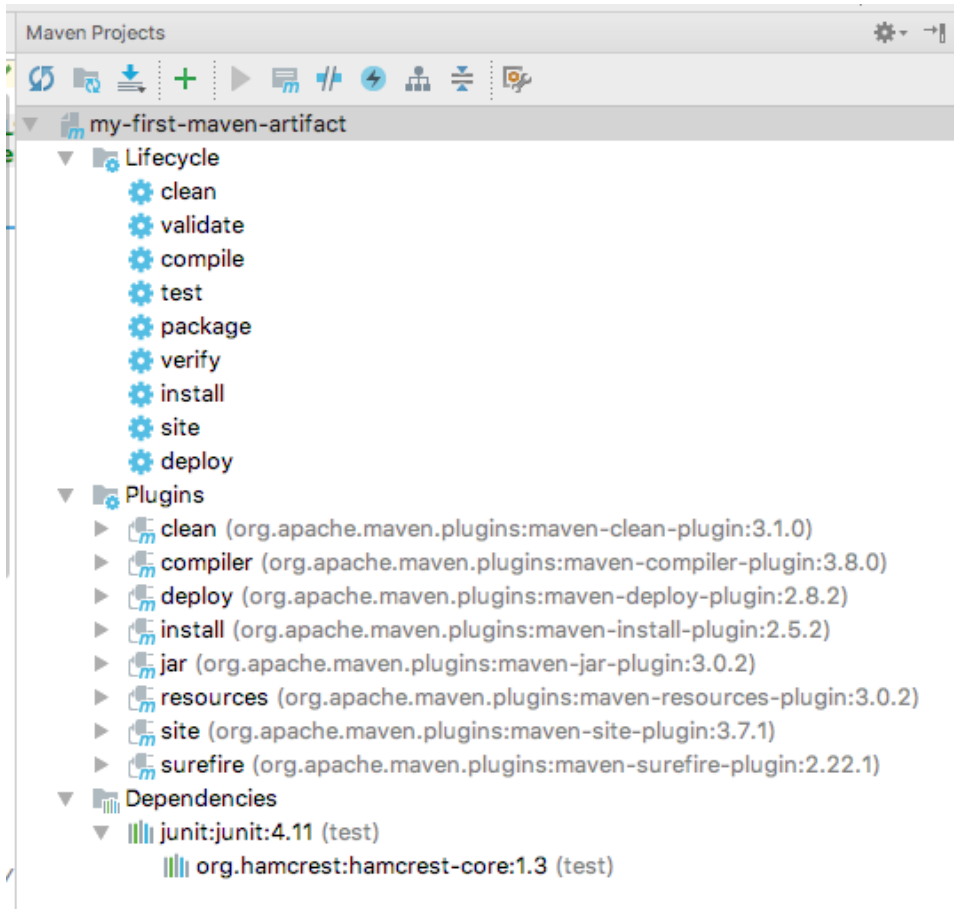
Maven - POM Editor (Text)

Neues Projekt, POM

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http
      ://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>de.nordakademie.pdse</groupId>
  <artifactId>my-first-artifact</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>groups.a</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.
        sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

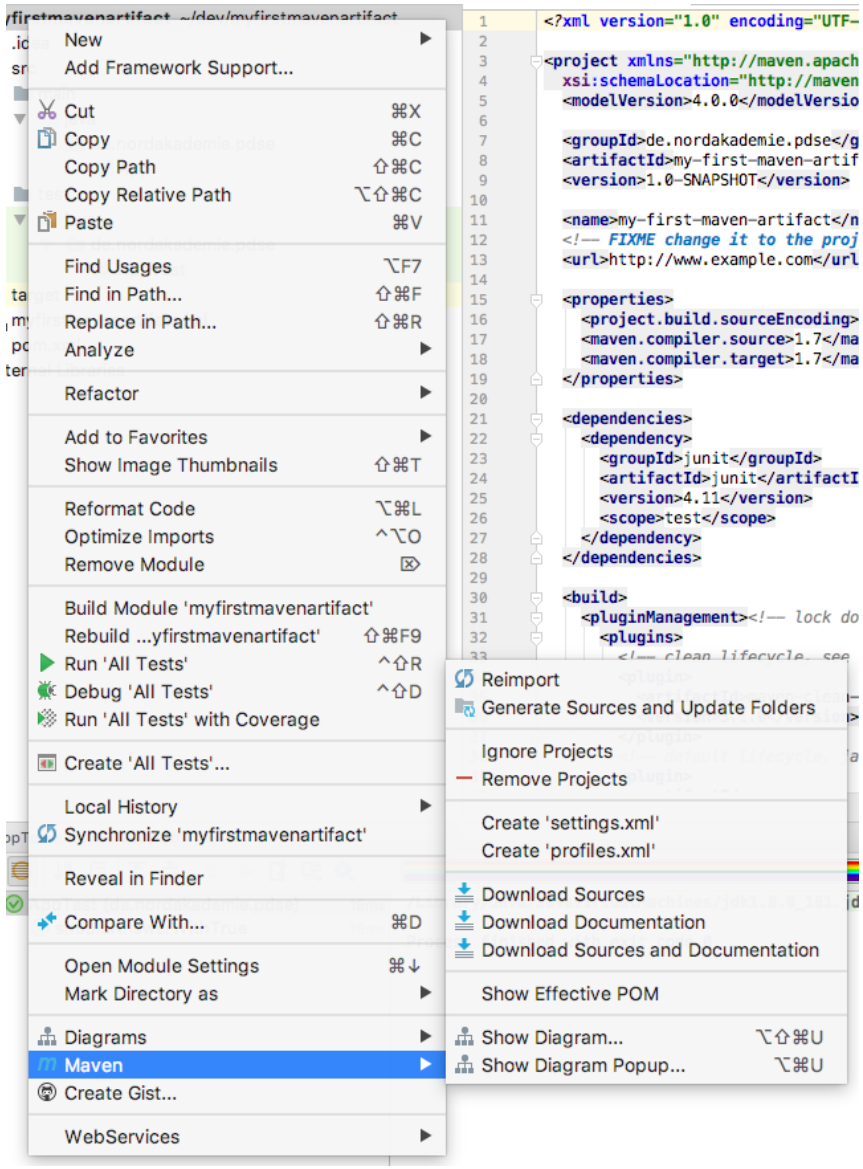
Maven

Maven IntelliJ Integration



Maven

Projektkontextmenü



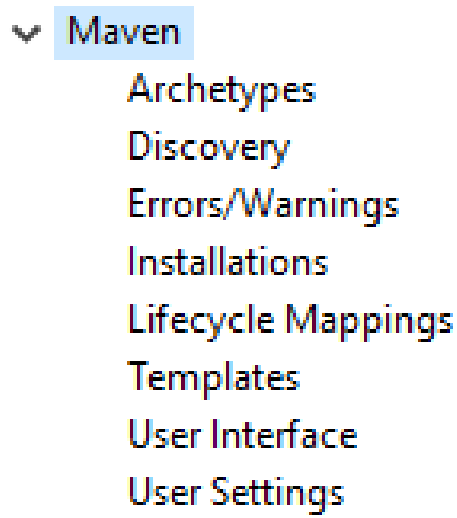
Exkurs: Maven in Eclipse

Exkurs: Maven in Eclipse

- Eclipse baut die Software „während“ man tippt / bzw. sobald man die Software starten möchte
- In Maven definierte Abhängigkeiten und die durch Maven Plug-Ins erreichten und in einem Projekt verwendeten Fähigkeiten müssen auch innerhalb von Eclipse ausgewertet werden. (Oder durch Verdopplung des Aufwandes in Eclipse soweit möglich nachgebaut werden).

Maven - Konfiguration

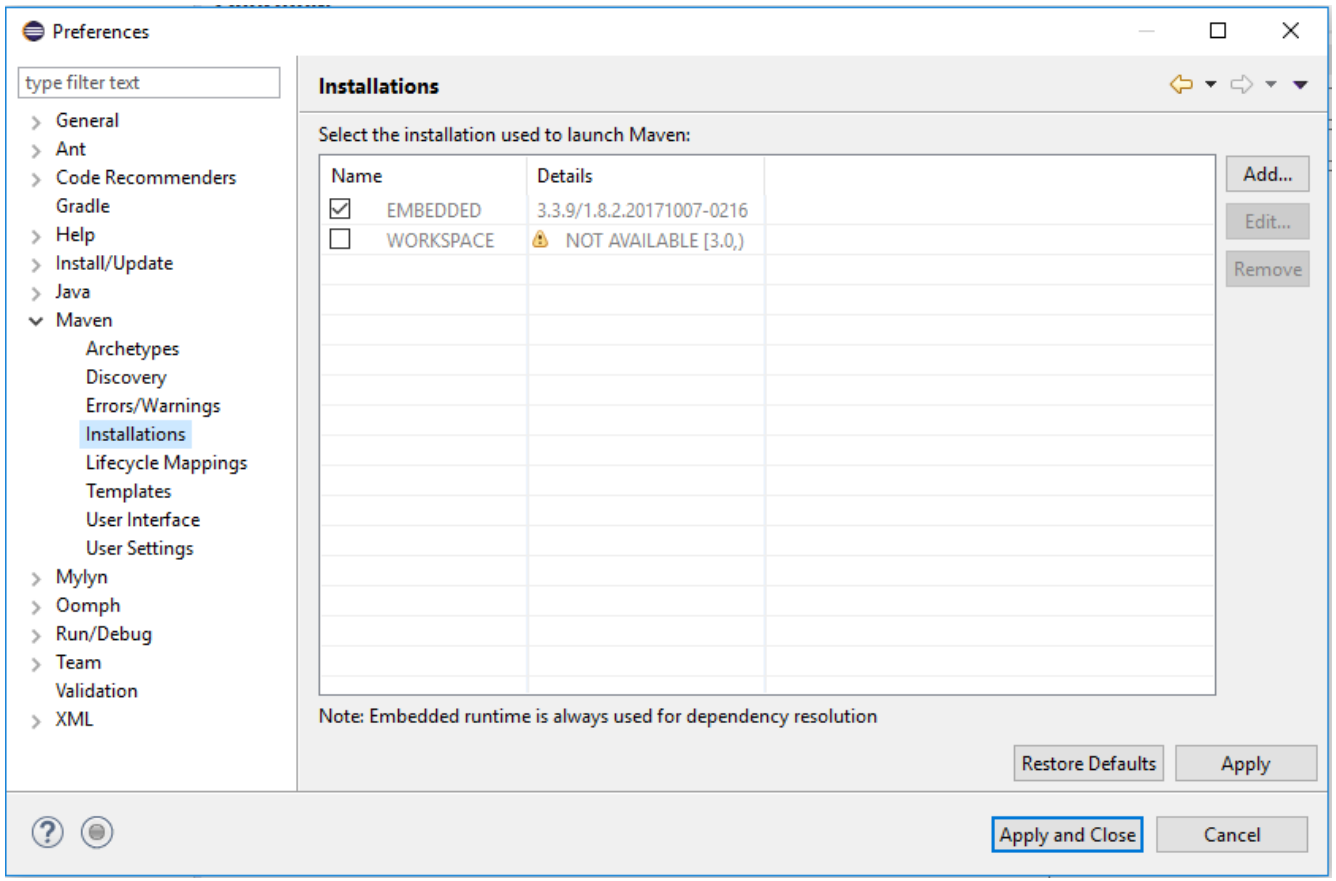
Windows → *Preferences* → *Maven*



Maven - Konfiguration

Maven Installations

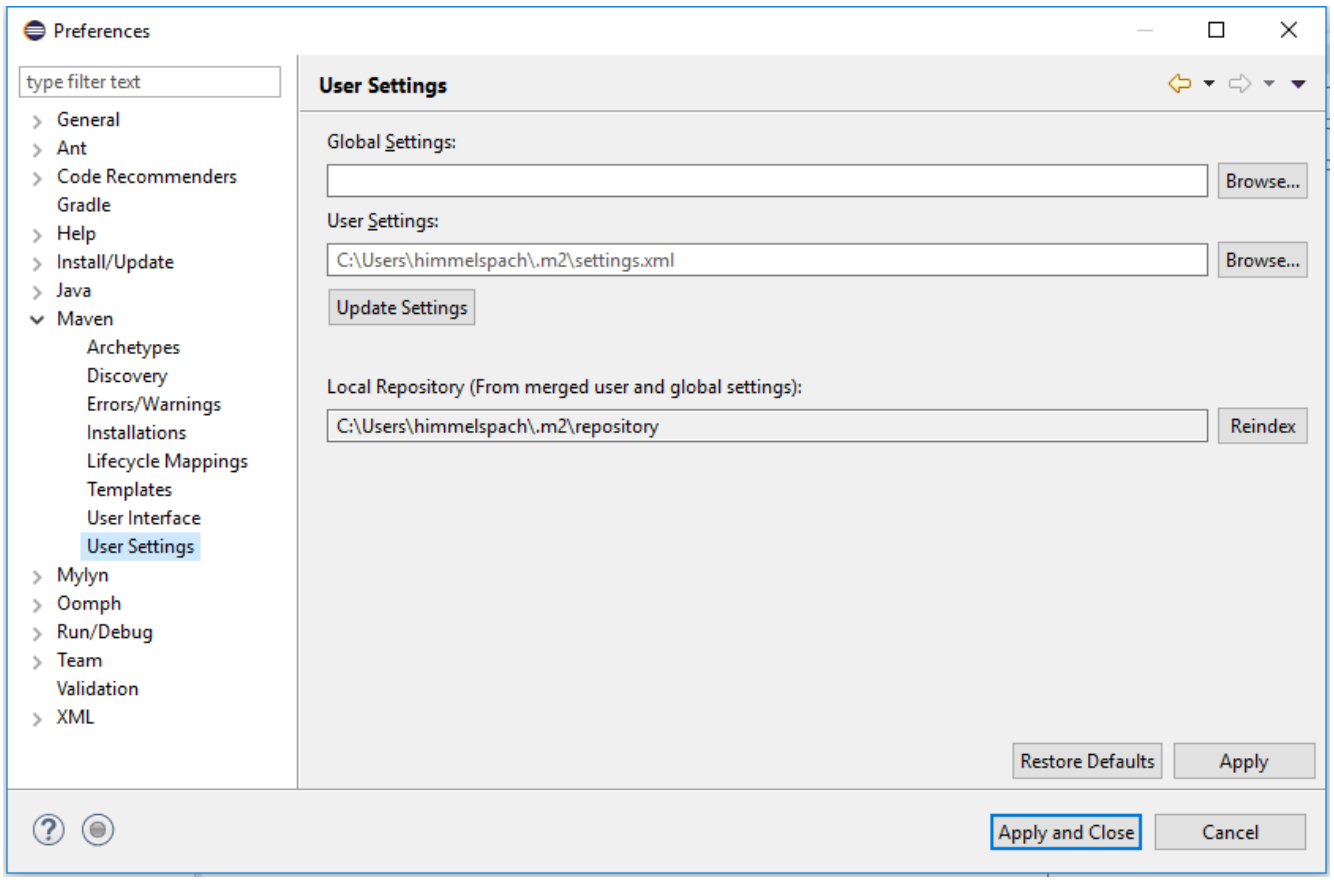
Windows → *Preferences* → *Maven*



Maven - Konfiguration

Maven Benutzereinstellungen

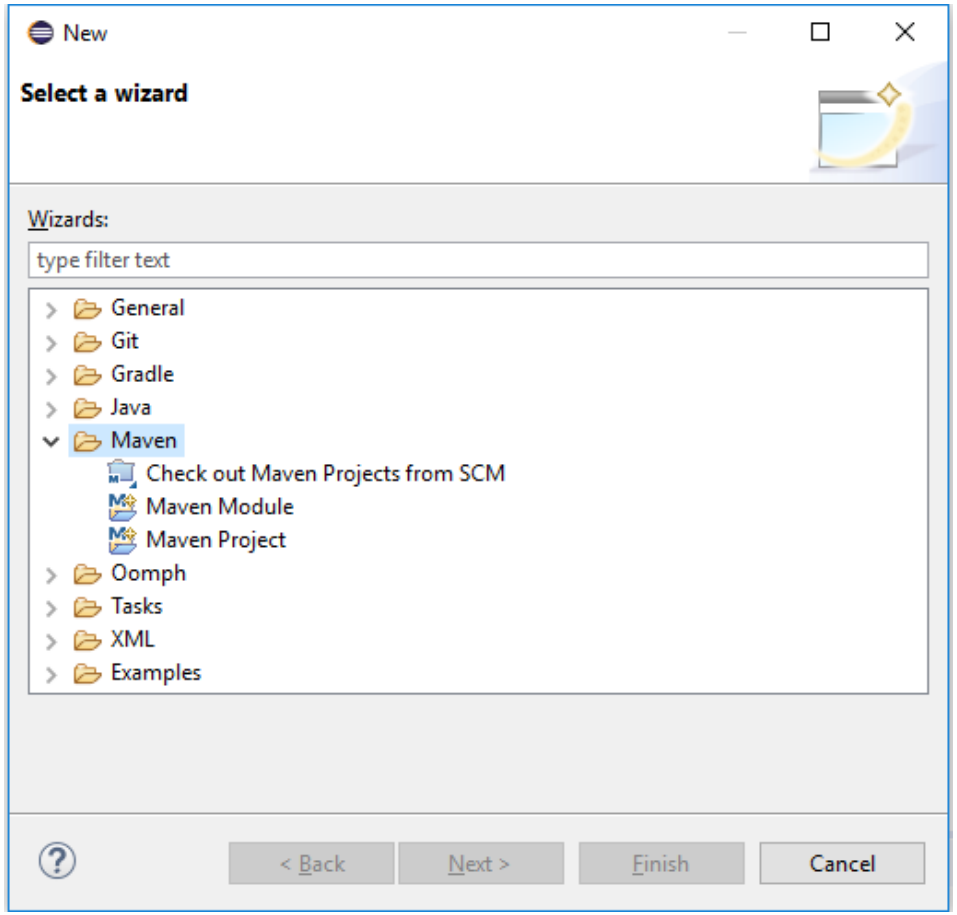
Windows → *Preferences* → *Maven*



Maven - Neues Projekt

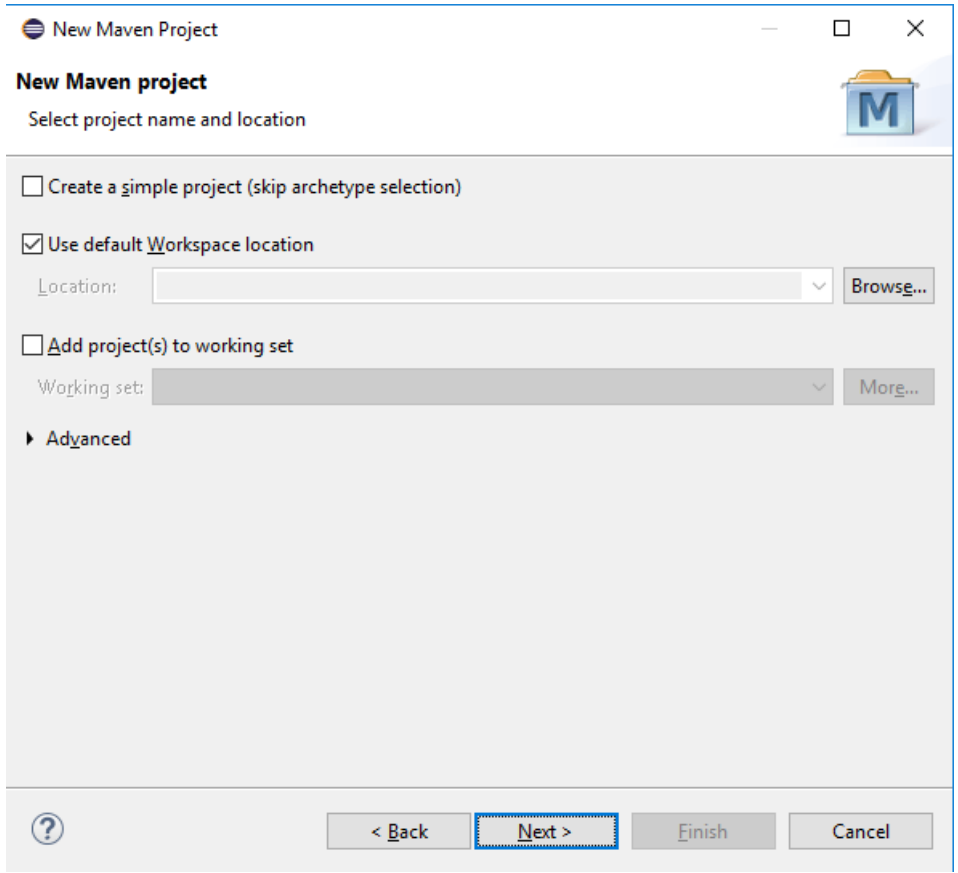
Auswahl eines Maven Projekts

File → *New* → *Other* → *Maven*



Maven - Neues Projekt

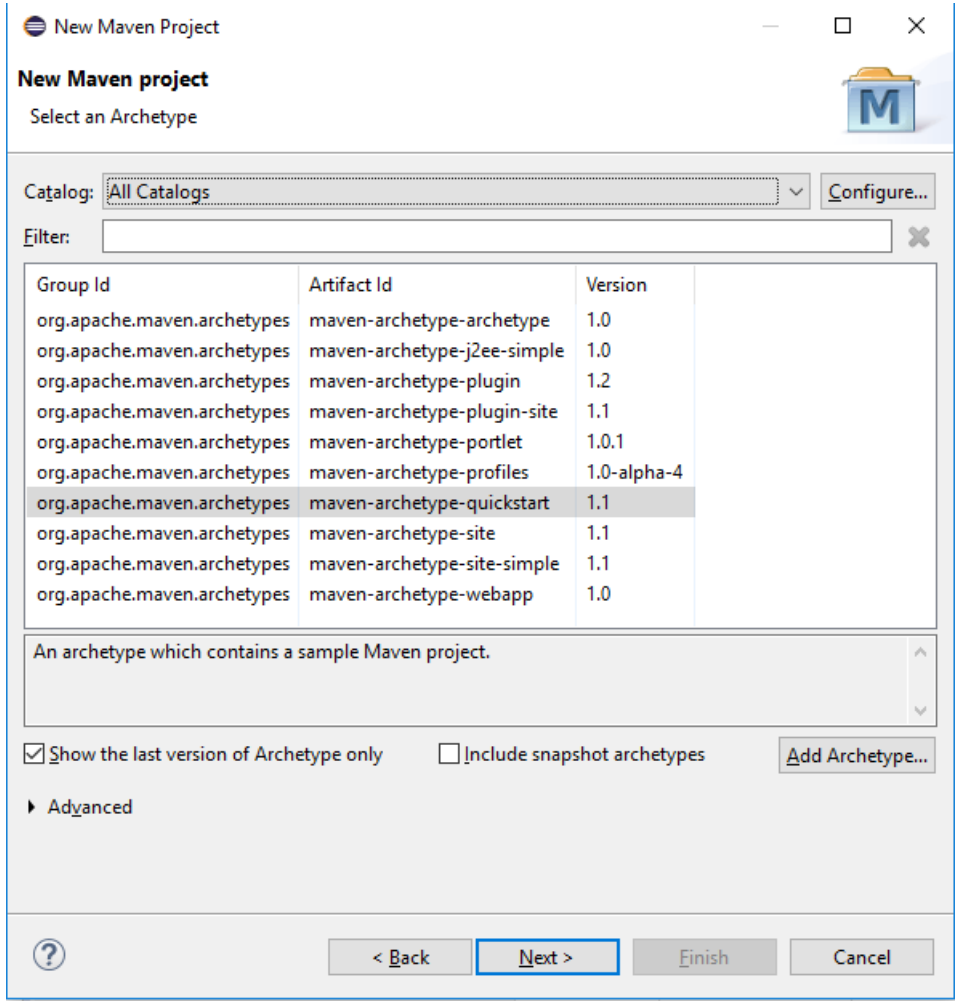
Ort des Projekts



Maven - Neues Projekt

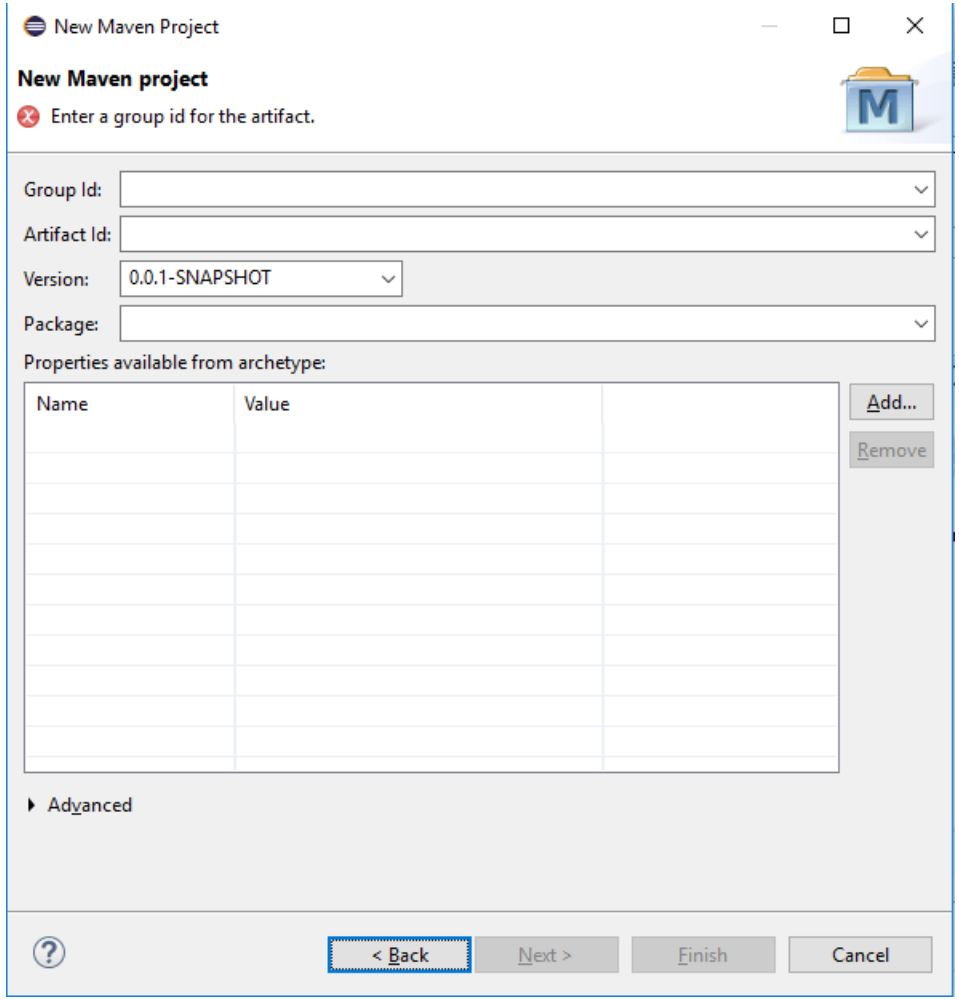
Auswahl einer Vorlage

Windows → *Preferences* → *Maven*



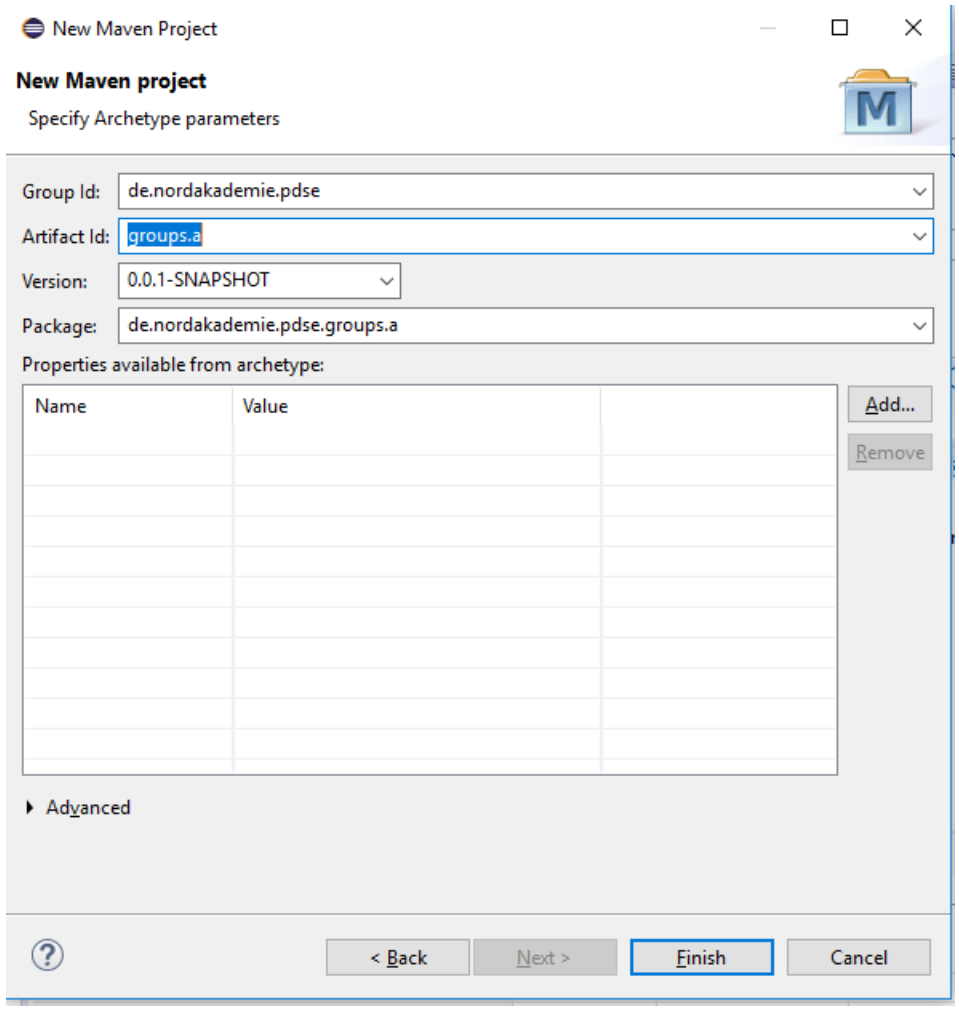
Maven - Neues Projekt

Identifikation des Projekts



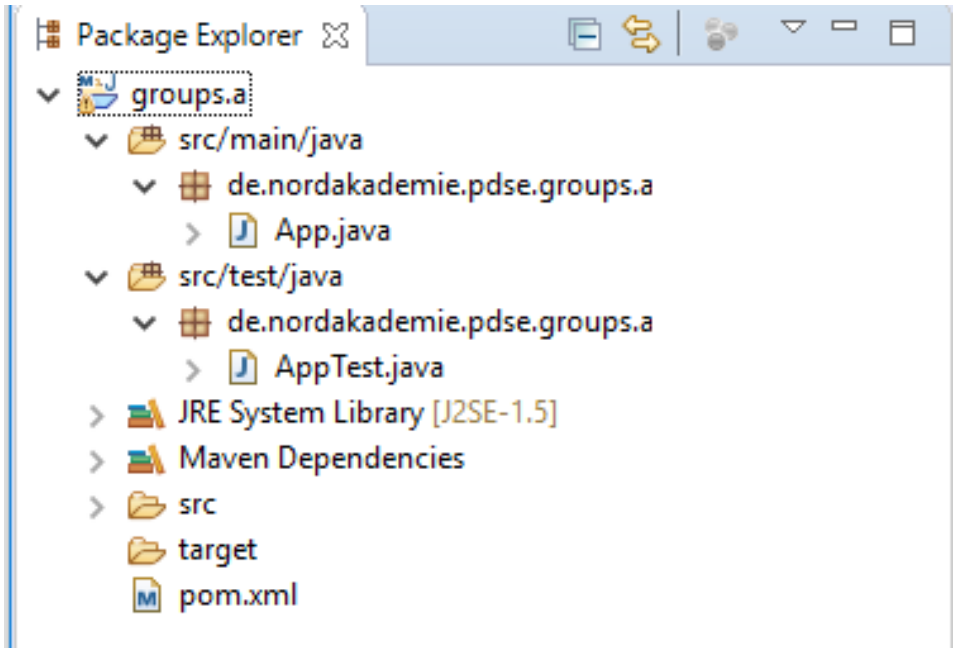
Maven - Neues Projekt

Identifikation des Projekts, Bsp



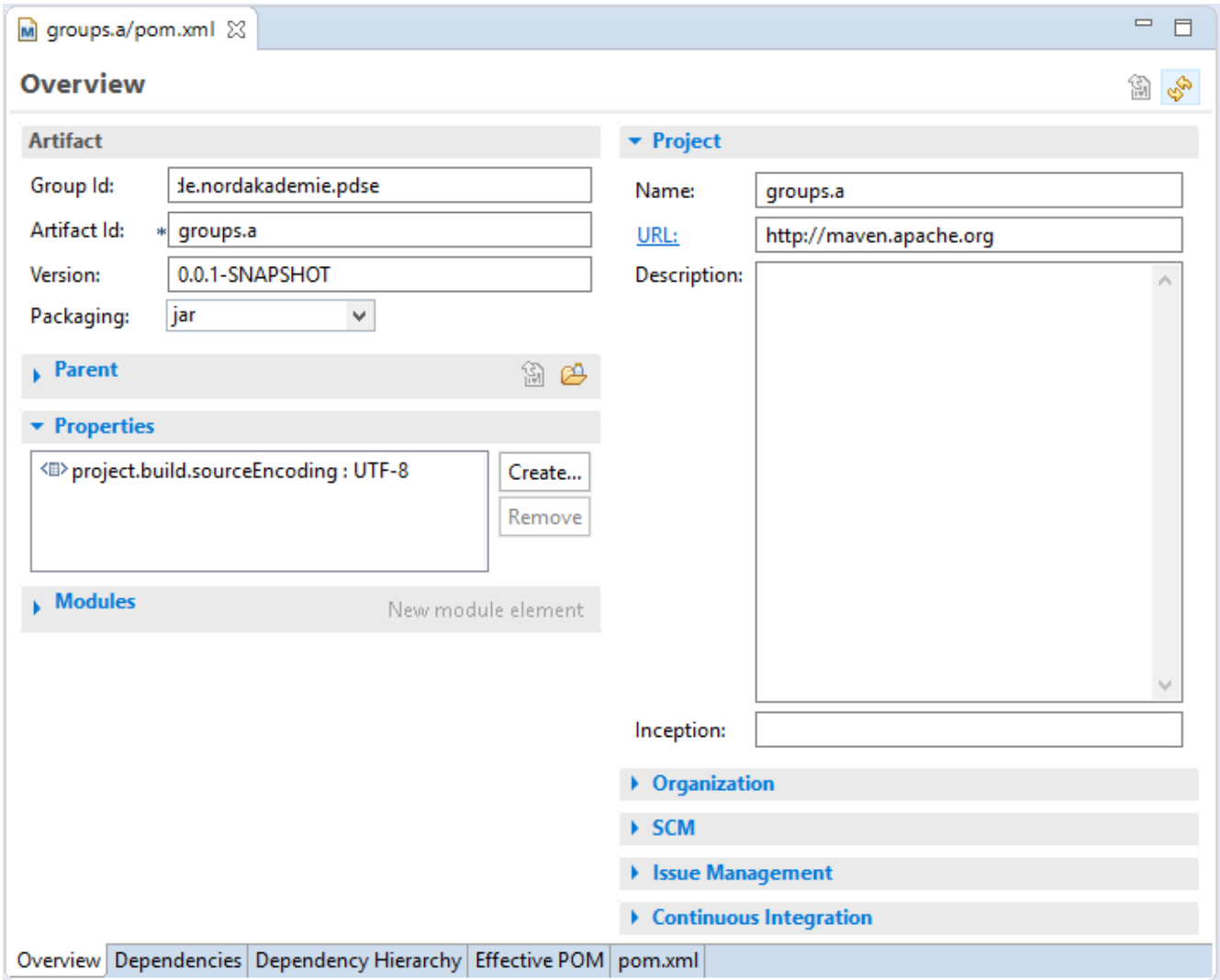
Maven - Neues Projekt

Erstellt, im Package Explorer offen



Maven - POM Editor (Grafisch)

Neues Projekt, POM



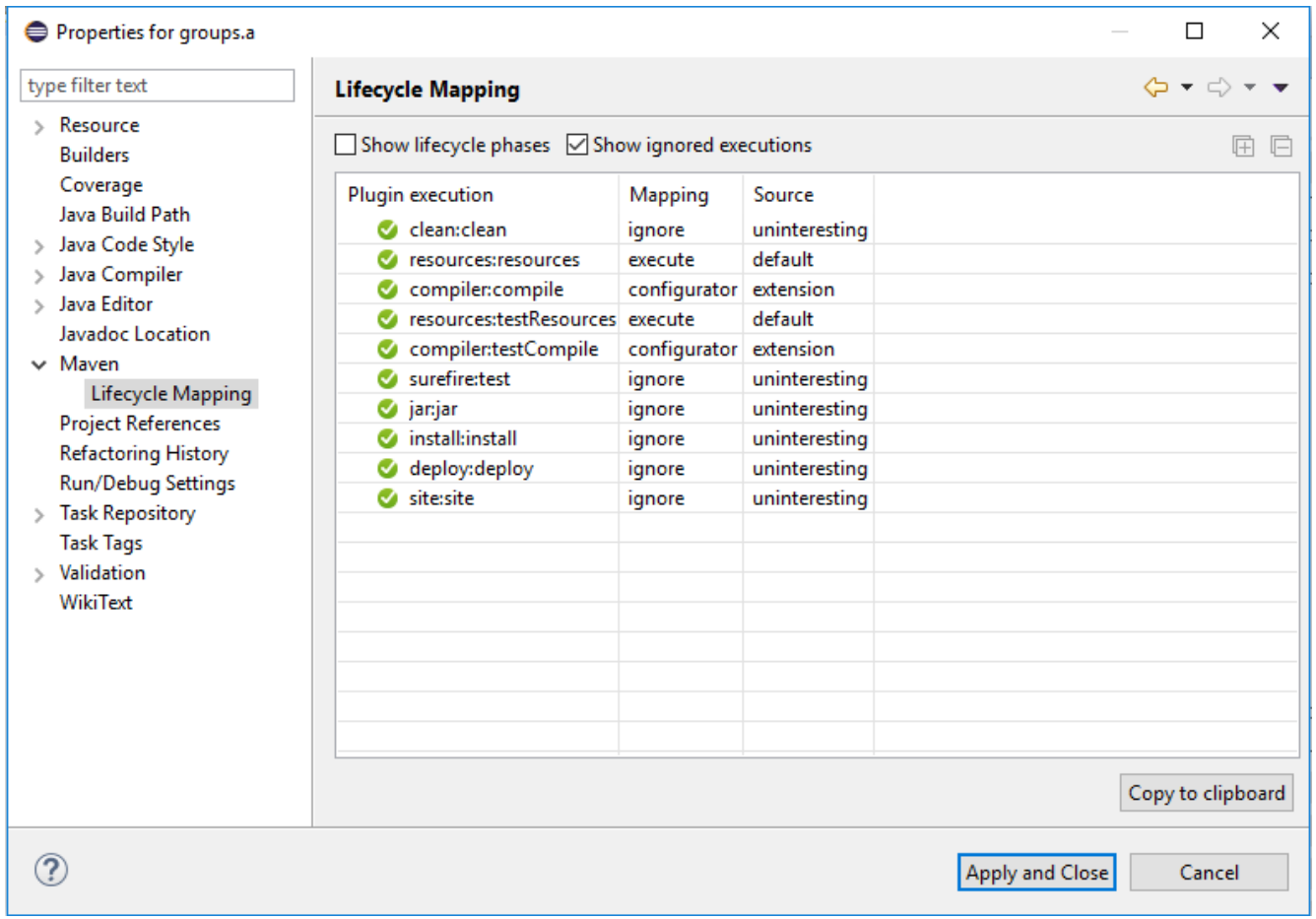
Maven - POM Editor (Text)

Neues Projekt, POM

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http
      ://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>de.nordakademie.pdse</groupId>
  <artifactId>groups.a</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>groups.a</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.
        sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

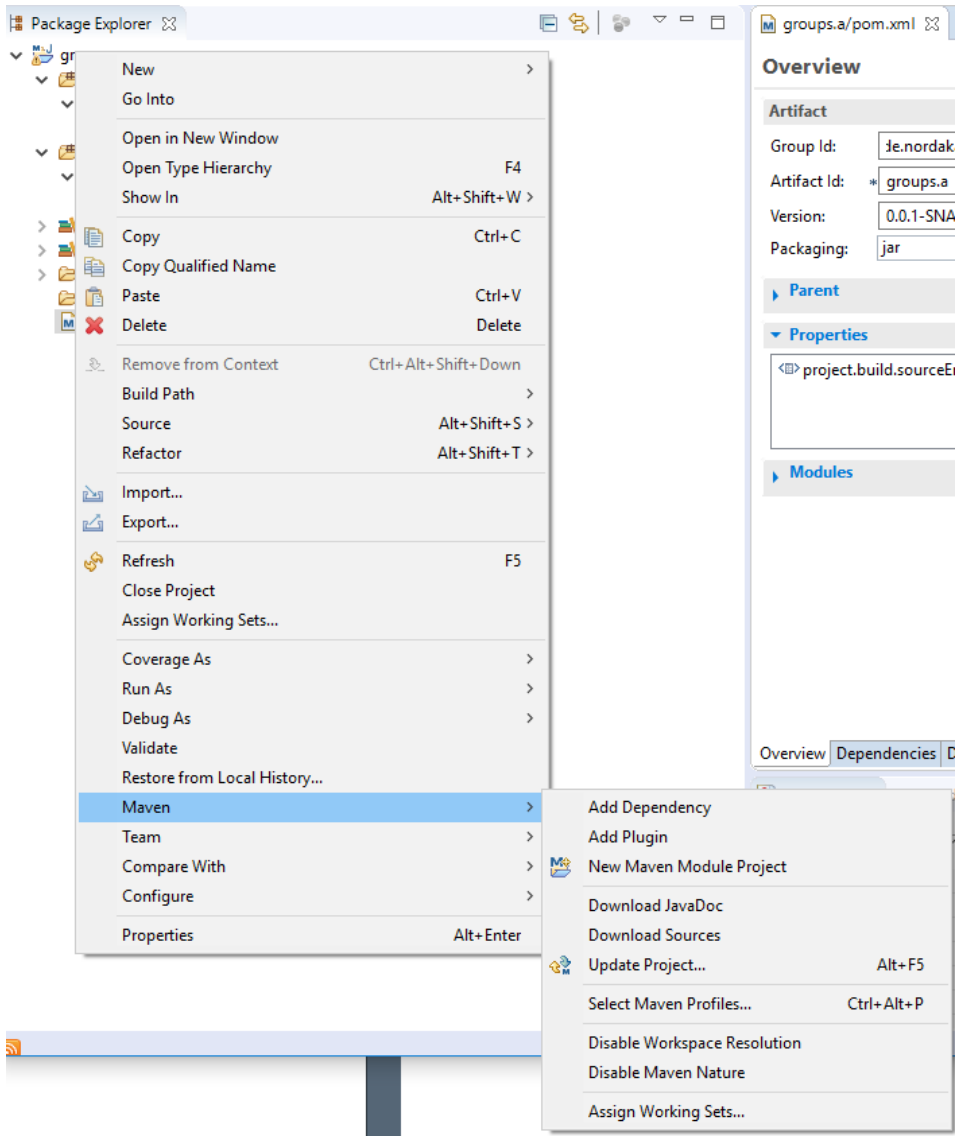
Maven

Projektspezifische Konfiguration



Maven

Projektkontextmenü



Aufgabe

Aufgabe 1/4

Erstellen Sie ein neues Maven-Projekt entsprechend des Foliensatzes

- Als Archetype verwenden Sie `maven-archetype-quickstart`, als ArtifactId ihren Nachnamen, als GroupId `de.nordakademie`.
- Erkunden Sie das Projekt im „Project Explorer“. Welche Verzeichnisse wurden erstellt?
- Erkunden Sie die pom.xml mit Hilfe Ihrer IDE. Schauen Sie sich auch die Effective POM an.
- Führen Sie den Befehl „mvn install“ aus (z.B. rechts via Maven > Lifecycle > install oder via Terminal mit angepasstem $PATH)
- Verifizeren Sie, ob der Unit Test fehlerfrei gelaufen ist und der Build erfolgreich war. Ggf. müssen Sie noch `<maven.compiler.source>17</maven.compiler.source>` und `<maven.compiler.target>17</maven.compiler.target>` in <properties> der pom.xml ergänzen.

Aufgabe 2/4

Im Maven-Projekt Libraries einbinden

- Fügen Sie eine Abhängigkeit zu gson in der aktuellen Version ein (scope: compile). Sie können dazu einen der Editoren oder aber einen speziellen Dialog (Kontextmenü) verwenden (hier gibt es ggf. weitere Informationen: `https://mvnrepository.com/`)
- Suchen und finden Sie das gson Jar auf dem Datenträger.
- Erstellen Sie eine Klasse `Person` mit den Exemplarvariablen `firstName` und `lastName` mit Getter/Setter-Methoden.
- Erzeugen Sie in der Main-Methode der Klasse `App` ein Exemplar dieser Klasse und setzen Sie diese beiden Variablen.
- Erstellen Sie ein Exemplar der importierten Klasse Gson und rufen Sie auf diesem Objekt `toJson()` auf. Geben Sie den Rückgabewert auf der Konsole aus.
- Überprüfen Sie mit dem Aufruf der Main-Methode die Konsolenausgabe.

Aufgabe 3/4

Im Maven - Projekt Build-Plugin einbinden

- Fügen Sie in der POM-Datei im Build-Tag den finalName `${project.groupId}-${project.artifactId}-${project.version}` ein. Die Folie `Einstellung für das Bauen der Software 1/4` hilft Ihnen. Starten Sie erneut Maven Install. Prüfen Sie im target-Ordner den Namen der JAR-Datei.
- Fügen Sie in der POM-Datei im Build-Tag unter Plugins das Plugin JaCoCo hinzu. Übernehmen Sie dazu die Plugin-Konfiguration unter „project“ > „build“ > „plugins“ von `https://www.jacoco.org/jacoco/trunk/doc/examples/build/pom.xml`. Passen Sie die Version des Plugins auf 0.8.11 (ohne Snapshot) an.
- Führen Sie Maven Install aus. JaCoCo prüft die Coverage Ihrer Unit Tests. Da keine Unit Tests den Quellcode testen, bricht der Build ab. Dieses Plugin stellt also sicher, dass nur Quellcode gebaut werden kann, der hinreichend getestet ist. Ergänzen Sie die Tests.

Aufgabe 4/4

Die Sanduhr mit Maven in eine ausführbare Datei verwandeln

- Starten Sie mit Ihrer Lösung von der Sanduhr oder checken Sie sich die Musterlösung aus dem gitlab aus.
- Wandeln Sie das bestehende Projekt in ein Maven-Projekt um und bauen Sie dieses.
- Machen Sie sich mit dem Maven-Compiler build-Plugin vertraut.
  - Setzen Sie das Encoding auf UTF-8
  - Setzen Sie Eingabe und Ausgabe Java-Version auf 17
- Bauen Sie das Projekt und führen es mit folgendem Befehl aus `java -cp <jar-Datei> de.nordakademie.sandglass.Main`
- Als Letztes importieren Sie das build-Plugin `maven-jar-plugin` in der aktuellsten Version und konfigurieren es so, dass der Classpath mit aufgenommen wird und als Main-Klasse Ihre Hauptklasse genommen wird.
- Hiernach sollte das jar mit `java -jar <jar-Datei>` ausführbar sein.