

# Programmieren II

## Exkurs: Apache Maven

Institut für Automation und angewandte Informatik

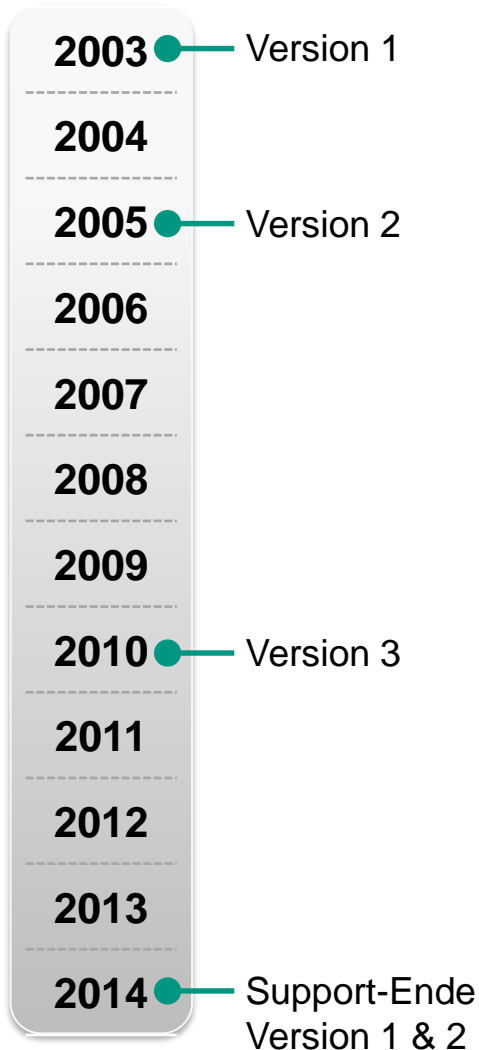
```
final List<String> allResults = new ArrayList<String>();  
final Map<String, Integer> typeWordResultCount = new HashMap<String, Integer>();  
final Map<String, Integer> typePoints = new HashMap<String, Integer>();  
evaluation.put(type, typePoints);  
  
for (final Sheet sheet : this.sheets) {  
    final String sheetResult = sheet.getPlayerInput(type);  
    if (sheetResult.startsWith(start) && this.isValidWord(sheetResult, type)) {  
        validWordCountForType++;  
        allResults.add(sheetResult);  
    }  
}
```

# Agenda

- Einführung in Apache Maven
  - Was ist Apache Maven?
  - Verwendung
  - Einbindung in IDE

# APACHE *ma*ven

# Entstehung



## ■ Maven 1

- Relativ hektische Umsetzung, daher einige Schwächen

## ■ Maven 2

- Konzept komplett überarbeitet
- Fehler von Maven 1 korrigiert
- Inkompatibel zu Maven 1

## ■ Maven 3

- Kompatibel zu Maven 2
- Ziel war Verbesserung der Benutzbarkeit und Stabilität

# Was ist Apache Maven

- „Maven ist ein Build-Management-Tool der Apache Software Foundation und basiert auf Java. Mit ihm kann man insbesondere Java-Programme standardisiert erstellen und verwalten.“

(Quelle: [https://de.wikipedia.org/wiki/Apache\\_Maven](https://de.wikipedia.org/wiki/Apache_Maven))

- Kommandozeilen-basiertes Tool

- Ziele

- Vereinfachung und Vereinheitlichung des Build-Prozesses
- Bereitstellung von Projektinformationen
- Förderung von „Best Practices“
- Transparentes Einbinden zusätzlicher Features mittels Plugins

# Vereinfachung und Vereinheitlichung des Build-Prozesses

- Die Details verschiedener Build-Mechanismen werden über vereinfachte Wrapper bereitgestellt
  - Struktur der Wrapper meist ähnlich, trotz sehr unterschiedlicher Tools die darunter angesprochen werden
- Projektbeschreibung mittels “Project Object Model” (POM)
  - Beschreibung der Abhängigkeiten
  - Maven-Projekte “funktionieren” alle gleich
  - Zusatzfunktionalität wird über Plugins standardisiert eingebunden

# Bereitstellung von Projektinformationen

- Standardisierte Informationen über Projekt
  - Grobe Zugehörigkeit zu Projektgruppe („groupId“)
  - Konkretes Projekt innerhalb der Gruppe („artifactId“)
  - Versionsnummer (inkl. Markierung von Entwicklungsversionen)
  - Optional möglich
    - Ausführlicher Name des Projekts
    - Beschreibungstext für das Projekt
- Klare Definition verwendeter Software durch zentral verwaltete Abhängigkeiten
- Direkte Einbindung von SCM-Systemen (Git, SVN, ..) vorgesehen
- Möglichkeit zur automatischen Generierung von Test-Reports

# Förderung von „Best Practices“

- Standardisierte Projektstruktur („Ordner-Struktur“)
  - Nicht mehr abhängig von der jeweiligen IDE
- Best Practices werden gefördert, jedoch nicht erzwungen
- Beispiel: Unit-Tests
  - Tests sind zentraler Bestandteil des Maven-Entwicklungszyklus‘
  - Test-Klassen werden in separatem, parallel zu den „normalen“ Klassen liegenden Quellcode-Baum erwartet
  - Namenskonventionen müssen eingehalten werden, damit Tests gefunden werden
  - Keine Custom-Umgebung für Tests
    - Test-Klassen müssen Test-Umgebung selbst initialisieren

# Transparentes Einbinden zusätzlicher Features

- Einfach gehaltenes Plugin-System für Erweiterung der Funktionalität
- Sauber geschriebene Maven-Plugins funktionieren für alle Software-Projekte
  - „Früher“: geskriptete Sonderlösungen meist für konkretes Projekt
- Plugin-Update kann Verbesserungen für alle verwendenden Projekte bereitstellen



# Abhängigkeitsverwaltung

- Wenn Software-Abhängigkeiten automatisch verwaltet werden, müssen diese „irgendwo“ verfügbar sein
- Maven verwendet hierfür Repositories
- Es gibt 2 Arten von Repositories
  - Lokal: Ordner auf dem Rechner mit dem Maven-Projekt. Verwaltet eigene Projekt-Artefakte und cacht verwendete Drittsoftware
  - Remote: Server, der bestimmten „Katalog“ von Software bereitstellt, die in Maven-Projekten genutzt werden kann

# Abhängigkeitsverwaltung II

- Abhängigkeiten werden *beschrieben*, nicht heruntergeladen und in das Projekt kopiert
  - Bibliotheken liegen nicht mehr in „Binär“-Form innerhalb des Software-Projekts
  - Das Projekt kann vollständig in ein Versionskontrollsystem geladen werden, ohne Fremd-Software dort mitzuverwalten

# Lebenszyklus eines Maven Projekts

- Maven bildet einzelne Schritte des Projekts in einem Lebenszyklus ab
- Häufig ausreichend: Standard-Lebenszyklus
  - Nicht jedes Software-Projekt muss jede der Phasen durchlaufen

Phase	Beschreibung
archetype	Erstellung des Software-Projekts, ggf. auf Basis eines zuvor definierten Templates
validate	Überprüfung der Projektstruktur auf Gültigkeit und Vollständigkeit
compile	Übersetzung des Quellcodes (z.B. .java → .class)

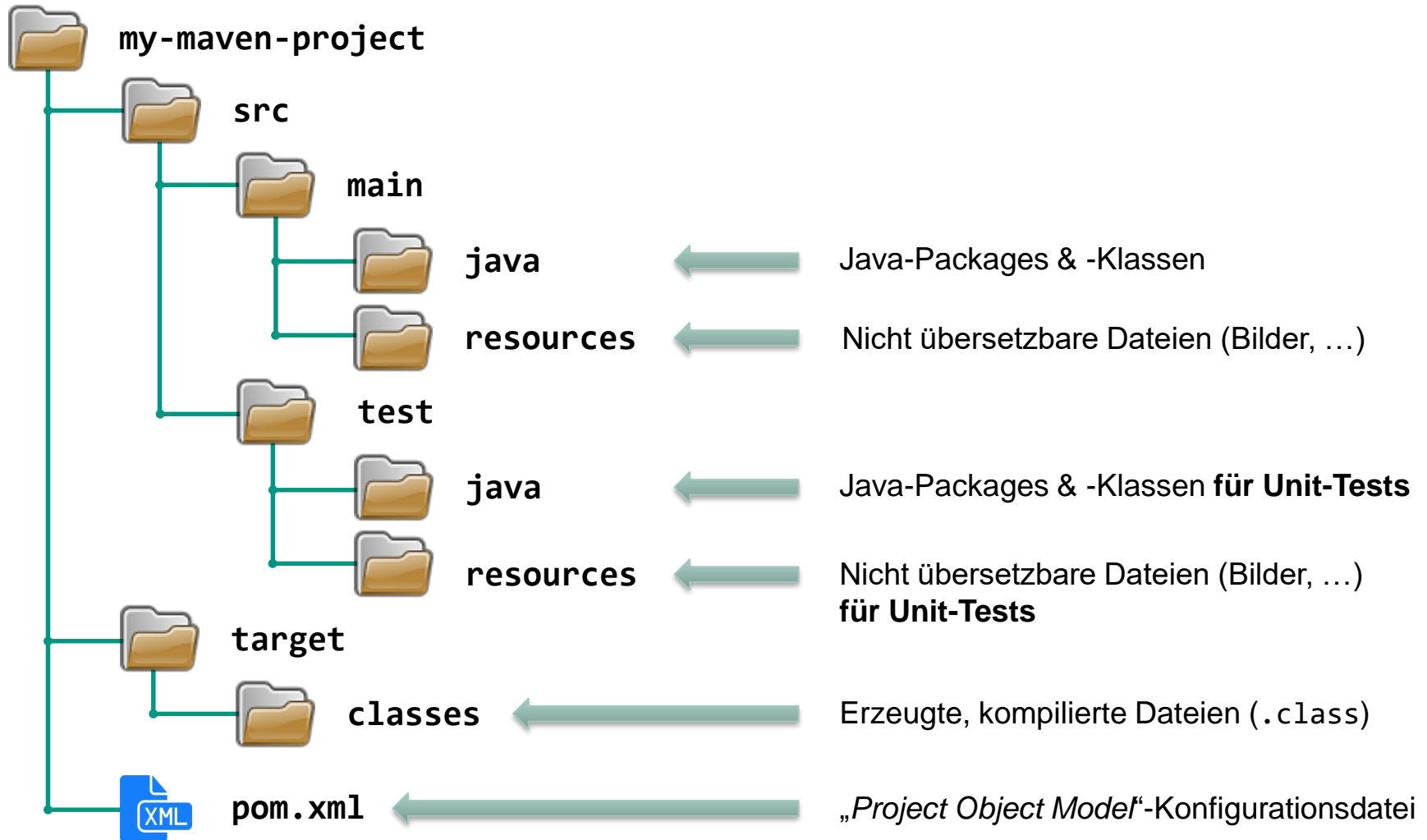
# Lebenszyklus eines Maven Projekts II

Phase	Beschreibung
test	Ausführung der Unit-Tests aus dem parallelen Quellcode-Baum mit passendem Test-Framework (z.B. JUnit)
package	Die übersetzten Dateien werden zum fertigen „Produkt“ verpackt, bspw. ein JAR-Archiv. Hier werden ggf. auch Dateien eingefügt die nicht kompilierbar sind (Bilder, XML-Dateien, ...)
integration-test	Fertiges Produkt kann in eine andere Umgebung übertragen werden und dort mit anderen Komponenten integriert getestet werden
verify	Überprüfung der Struktur des erzeugten Softwarepakets, ggf. Überprüfung von Qualitätskriterien
install	Installation im lokalen Maven-Repository
deploy	Installation in einem entfernten Maven-Repository

# Voraussetzungen

- Maven basiert auf Java und soll Software erzeugen
  - ➔ installiertes JDK nötig
  - ➔ Umgebungsvariable \$JAVA\_HOME muss gesetzt sein
  
- Variante 1: Maven herunterladen und installieren
  - Download von <http://maven.apache.org>
  - Installation besteht aus Entpacken & Anpassen der Umgebungsvariable \$PATH
  
- Variante 2: IDE-Integration
  - m2e-Plugin für Eclipse (in neueren Eclipse-Versionen bereits enthalten)
  - NetBeans & IntelliJ bringen ebenfalls bereits Plugins mit

# Projektstruktur



# Project Object Model – `pom.xml`

- Zentrale Konfigurationsdatei für Maven-Projekt
- Definiert und instrumentiert zuvor genannte Funktionen
  - Projektinformationen
  - Abhängigkeiten zu anderen Software-Bibliotheken
  - Verwendung und Konfiguration zusätzlicher Plugins
  - Angabe von Repositories
- `pom.xml`-Datei wird auf Wohlgeformtheit und Gültigkeit zu Beginn jedes Maven-Aufrufs geprüft

# Project Object Model – pom.xml II

## ■ Standard-Konfigurationsdatei

```
<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.dhbwka.java.exercise</groupId>
```

Gruppenzugehörigkeit, meist  
Java-Package

```
  <artifactId>my-maven-project</artifactId>
```

Projekt-Identifizier

```
  <version>0.0.1-SNAPSHOT</version>
```

Aktuelle Versionsnummer

```
  <name>Maven Demo Project</name>
```

Name des Projekts

```
  <packaging>jar</packaging>
```

Typ des resultierenden  
Software-Artefakts

```
</project>
```



# pom.xml Abhängigkeitsverwaltung

- Abhängigkeiten („Dependencies“) verwalten
  - Google GSON via Maven einbinden

```
<project xmlns="..." xmlns:xsi="..." xsi:schemaLocation="...">
```

(...)

```
<dependencies>  
  <dependency>  
    <groupId>com.google.code.gson</groupId>  
    <artifactId>gson</artifactId>  
    <version>2.8.5</version>  
  </dependency>  
</dependencies>
```

(...)

```
</project>
```

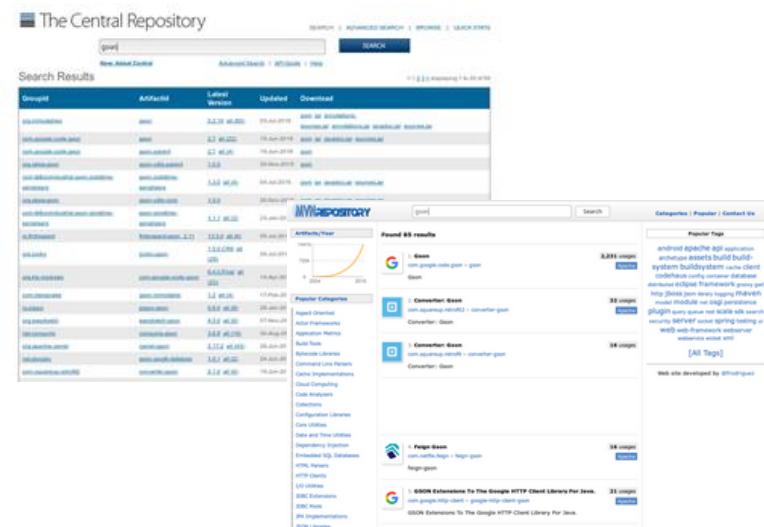
# pom.xml Abhängigkeitsverwaltung II

## ■ Gibt es ein Verzeichnis verfügbarer Bibliotheken?

- Bibliotheken aus Standard-Repository immer verfügbar
- Fremd-Repositories einbindbar

## ■ Übersicht über Standard-Repository

- <http://search.maven.org/>  
Standard-Suchmaschine für zentrales Maven-Repo
- <http://mvnrepository.com/>  
Alternative Suchmaschine für zentrales Maven-Repo



# pom.xml Abhängigkeitsverwaltung III

Home » com.google.code.gson » gson » 2.8.5



**Gson » 2.8.5**

Gson

License	Apache 2.0
Categories	JSON Libraries
Date	(May 22, 2018)
Files	jar (235 KB) View All
Repositories	Central Mulesoft
Used By	10,406 artifacts

Maven Gradle SBT Ivy Grape Leiningen Buildr

```
<!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.8.5</version>
</dependency>
```

```
<project xmlns="..." xmlns:xsi="..." xsi:schemaLocation="...">
```

(...)

```
<dependencies>
```

```
<dependency>
```

```
<groupId>com.google.code.gson</groupId>
```

```
<artifactId>gson</artifactId>
```

```
<version>2.8.5</version>
```

```
</dependency>
```

```
</dependencies>
```

(...)

```
</project>
```

# pom.xml Plugin-Konfiguration

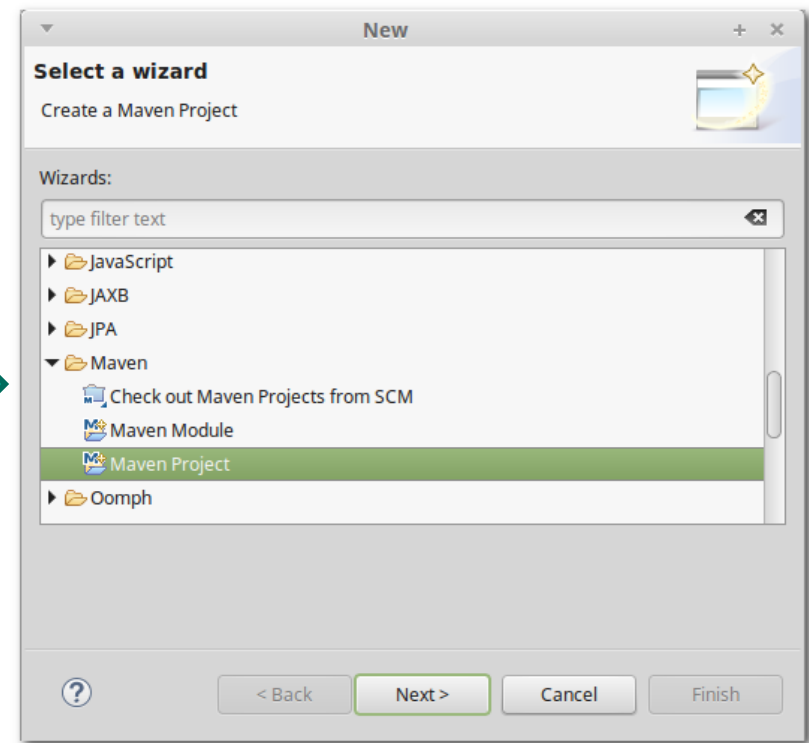
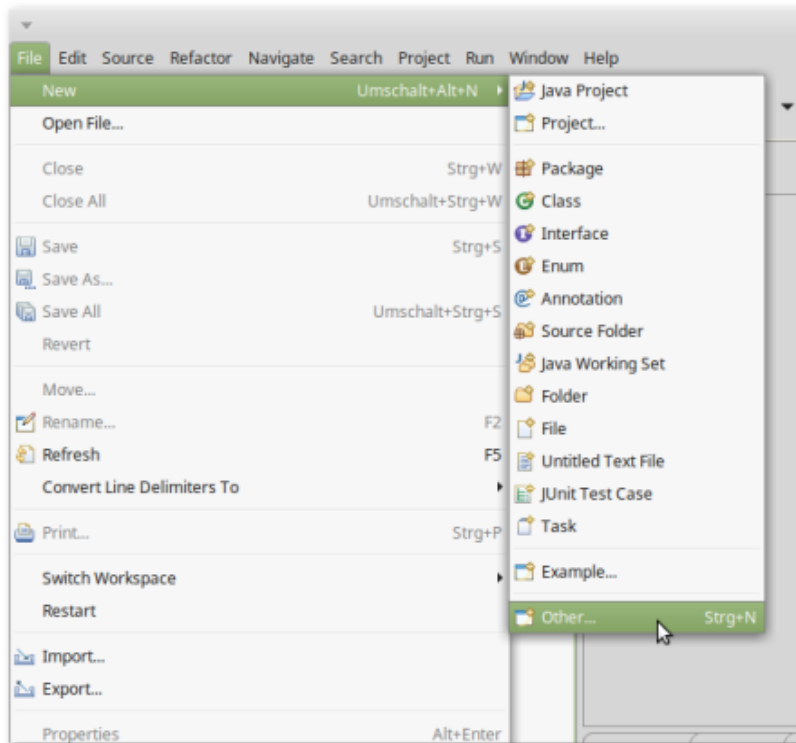
## ■ Konfigurieren der Java-Version des Projekts

```
<project xmlns="..." xmlns:xsi="..." xsi:schemaLocation="...">  
(...)  
  
  <properties>  
    <maven.compiler.source>1.8</maven.compiler.source>  
    <maven.compiler.target>1.8</maven.compiler.target>  
  </properties>  
  
(...)  
  
</project>
```

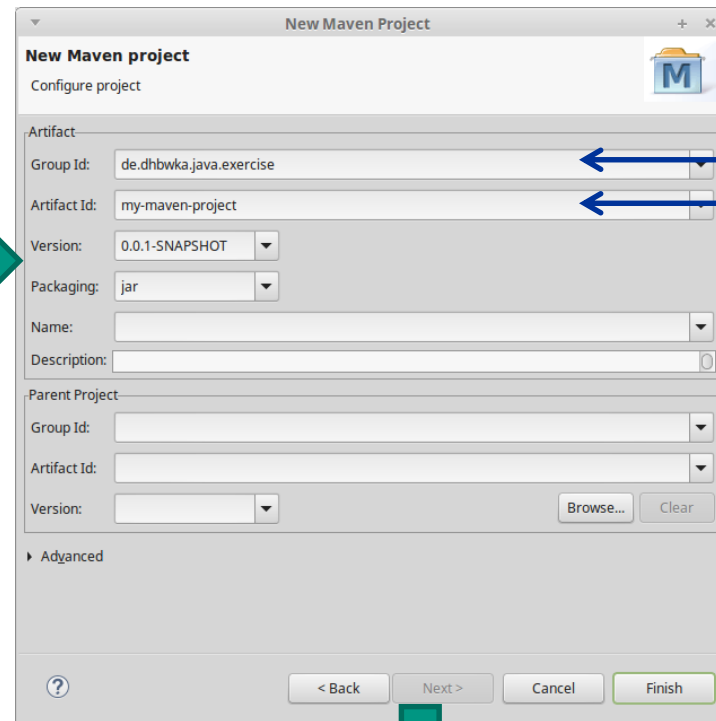
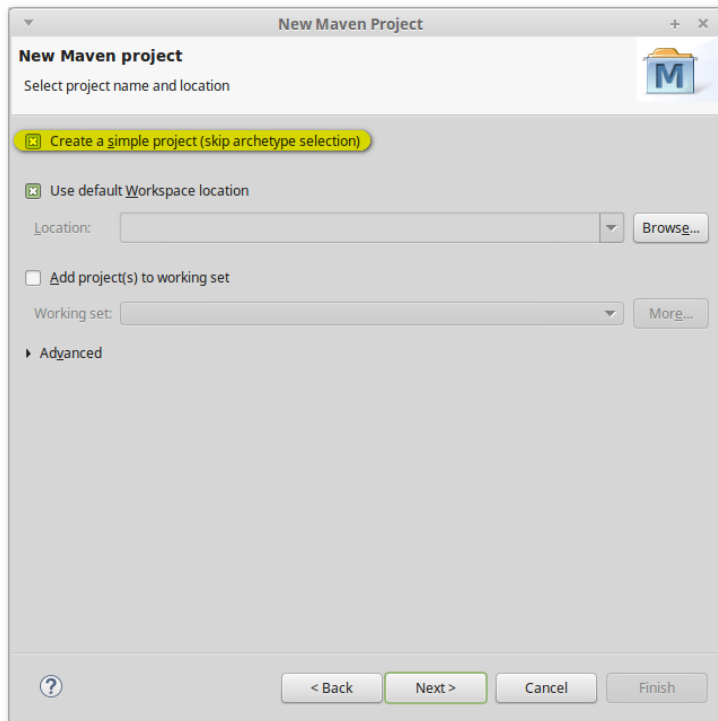
# Maven-Projekt erstellen

- Grundsätzlich: Maven ist ein Kommandozeilenwerkzeug
  - Projekt kann über Kommandozeile erstellt werden
  - Breite, UI-unterstützte Integration in IDEs
- Neue Projekte können basierend auf Projektschablonen („Archetypes“) erstellt werden
  - Wir nutzen zunächst eine „nackte“ Schablone

# Maven-Projekt erstellen: Eclipse

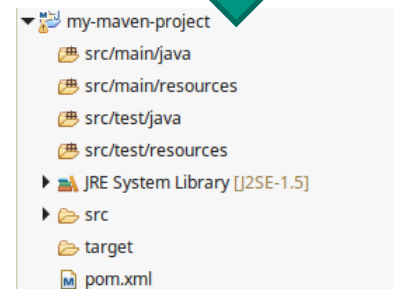


# Maven-Projekt erstellen: Eclipse II

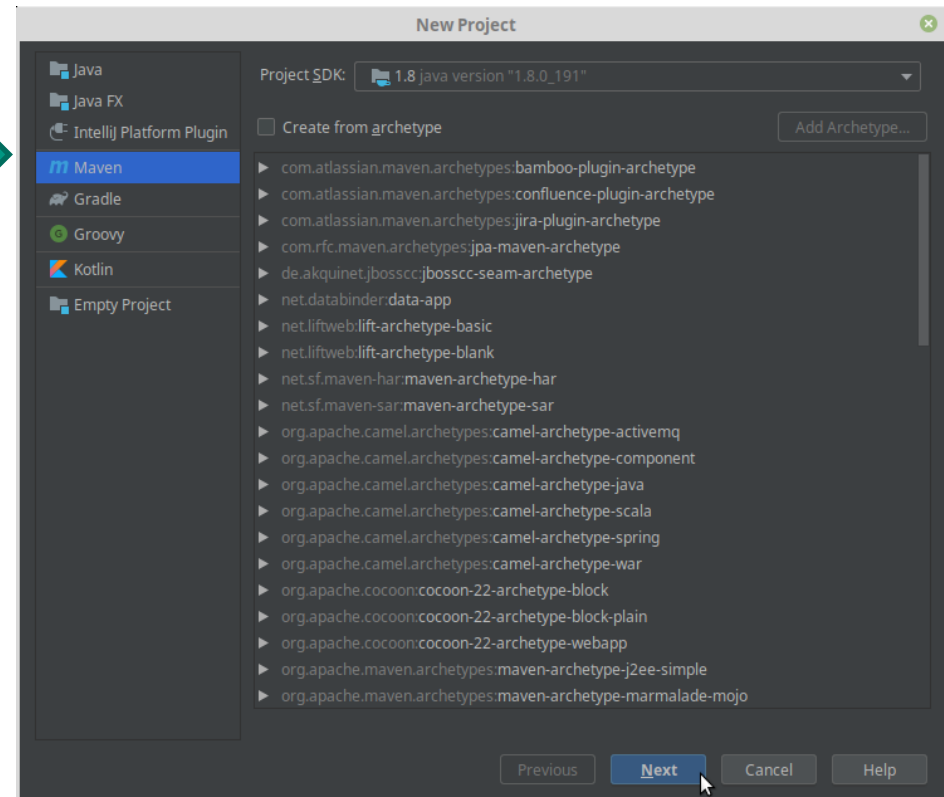
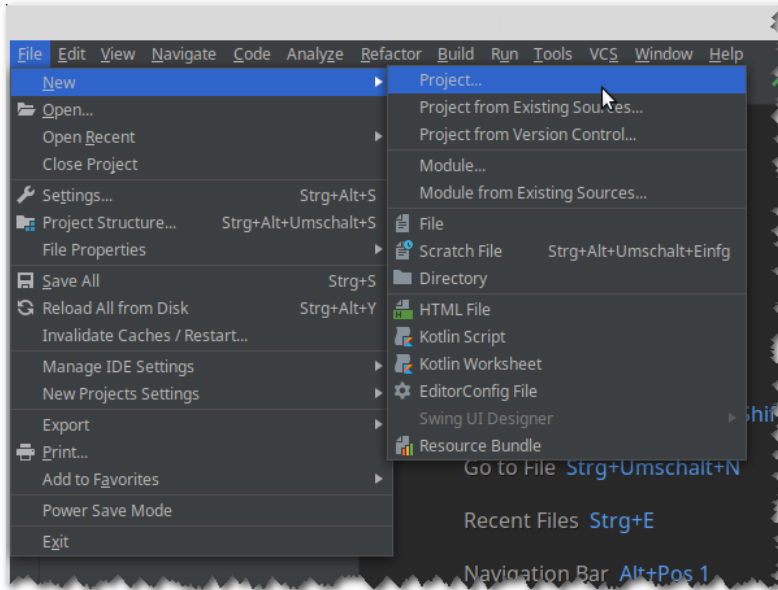


← Projektgruppe

← Projekt-ID



# Maven-Projekt erstellen: IntelliJ

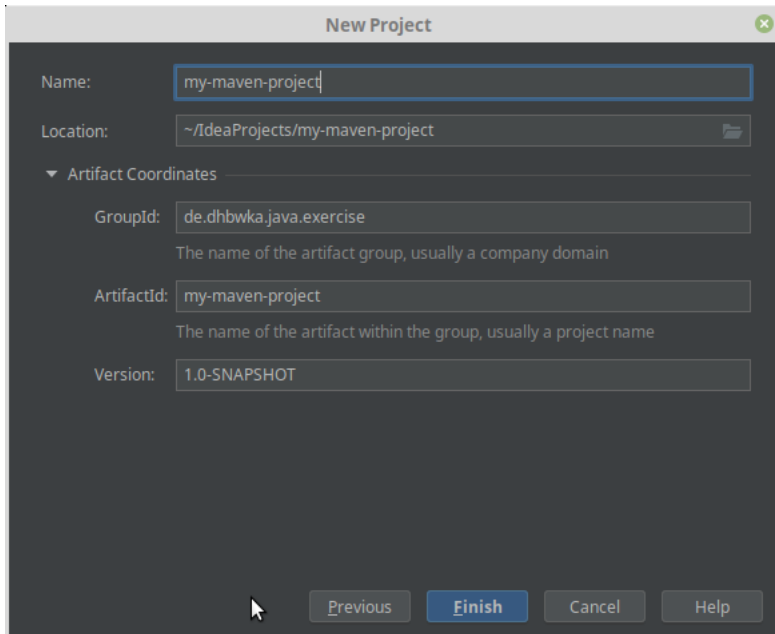


IntelliJ IDEA

**maven**



# Maven-Projekt erstellen: IntelliJ II



**New Project**

Name:

Location:

▼ Artifact Coordinates

GroupId:

The name of the artifact group, usually a company domain

ArtifactId:

The name of the artifact within the group, usually a project name

Version:

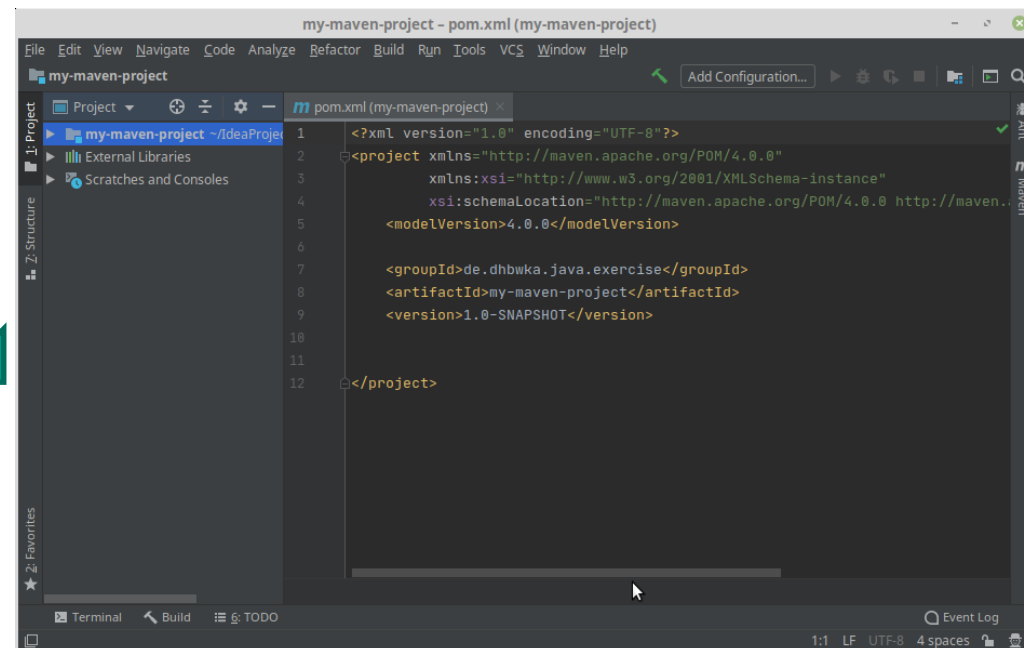
Previous Finish Cancel Help

← IntelliJ-Projektname

← Projektgruppe

← Maven-Projekt-ID

*kann identisch sein, muss nicht*



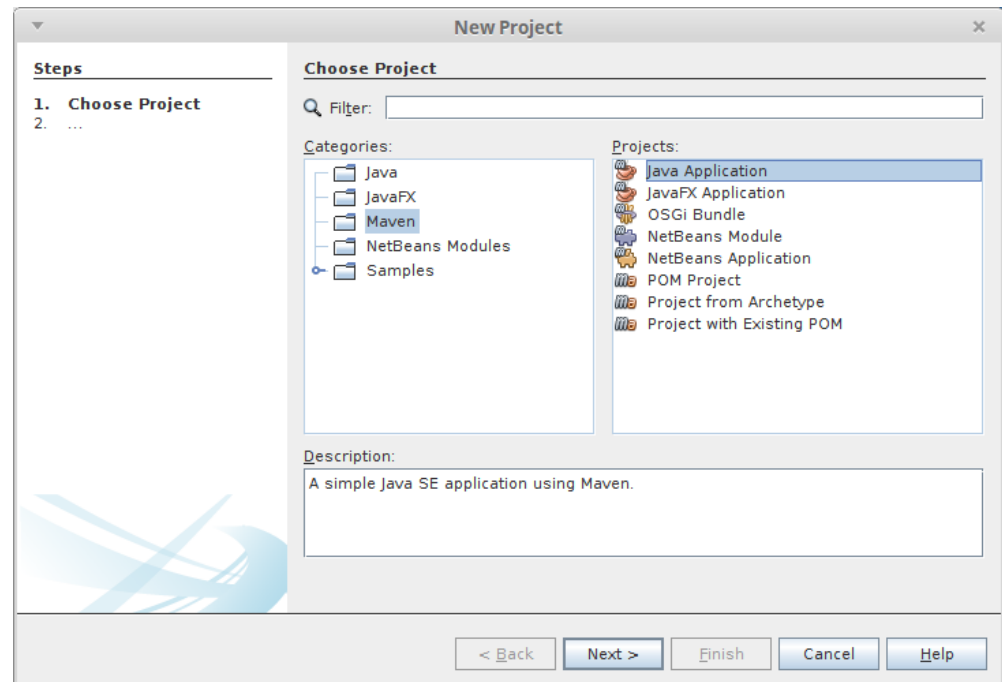
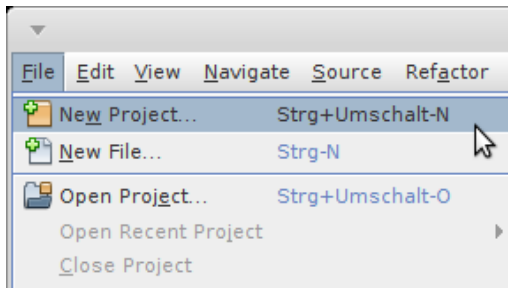
my-maven-project - pom.xml (my-maven-project)

```

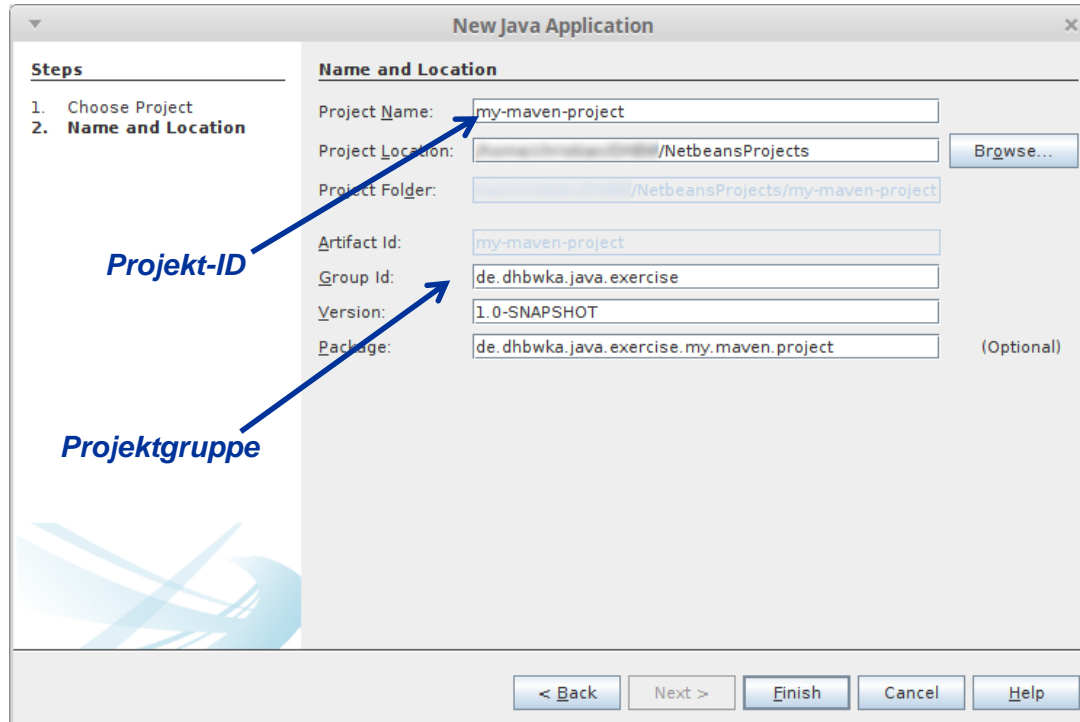
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
5      <modelVersion>4.0.0</modelVersion>
6
7      <groupId>de.dhbwka.java.exercise</groupId>
8      <artifactId>my-maven-project</artifactId>
9      <version>1.0-SNAPSHOT</version>
10
11
12  </project>

```

# Maven-Projekt erstellen: NetBeans



# Maven-Projekt erstellen: NetBeans II



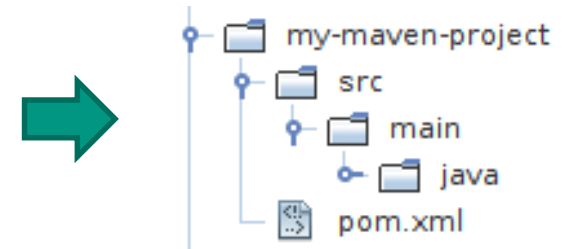
The dialog box 'New Java Application' is shown with the 'Steps' pane on the left and the 'Name and Location' pane on the right. The 'Steps' pane shows '1. Choose Project' and '2. Name and Location'. The 'Name and Location' pane contains the following fields:

- Project Name: my-maven-project
- Project Location: /NetbeansProjects (with a 'Browse...' button)
- Project Folder: /NetbeansProjects/my-maven-project
- Artifact Id: my-maven-project
- Group Id: de.dhbwka.java.exercise
- Version: 1.0-SNAPSHOT
- Package: de.dhbwka.java.exercise.my.maven.project (Optional)

Two blue arrows point from labels to the 'Group Id' and 'Artifact Id' fields:

- A blue arrow points from the label 'Projekt-ID' to the 'Artifact Id' field.
- A blue arrow points from the label 'Projektgruppe' to the 'Group Id' field.

At the bottom of the dialog are buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.



# Aufgabe – Erstes Maven-Projekt

- Erstellen Sie ein Maven-Projekt
  - Abhängigkeit zu Google GSON
  - Konfigurieren der Java-Version auf Java 1.8 (oder höher)
- ➔ Kopieren Sie den Code Ihres GSON-Projekts in den Quellcode-Ordner des Maven-Projekts und überprüfen Sie ob noch alles funktioniert!