Lernskript: Einführung in Maven

Maven ist ein Build-Automatisierungs- und Projektmanagement-Werkzeug, das drei entscheidende Probleme löst: die Verwaltung externer Bibliotheken, die Automatisierung des Build-Prozesses und die Sicherstellung einer standardisierten Projektstruktur.

Teil 1: Die grundlegende Analogie - Das "Warum"

Stell dir Maven als einen **LEGO-Baumeister-Roboter** vor. Die zentrale Konfigurationsdatei `pom.xml` (Project Object Model) ist sein Bauplan.

- **Abhängigkeitsmanagement:** Anstatt manuell nach LEGO-Steinen (Bibliotheken, `.jar`-Dateien) zu suchen, deklarierst du im `pom.xml`, was du brauchst. Maven lädt es automatisch und in der exakt richtigen Version aus dem zentralen Lager (Maven Central Repository).
- **Build Lifecycle:** Anstatt ein Modell von Hand zu bauen, gibst du dem Roboter Befehle (z.B. `mvn package`). Er folgt dann einem festen, standardisierten Prozess: Code validieren, kompilieren, testen und das Ergebnis in eine Schachtel (`.jar`-Datei) verpacken.
- **Konsistenz:** Jeder, der denselben Bauplan (`pom.xml`) und denselben Roboter (Maven) hat, baut am Ende ein exakt identisches Modell.

Teil 2: Praktische Anwendungsfälle - Das "Wofür"

Use Case 1: Abhängigkeiten verwalten

Das Herzstück von Maven. Man deklariert externe Bibliotheken im `<dependencies>`-Block der `pom.xml`.

- **Beispiel (Ausschnitt aus `pom.xml`):**

```
<dependencies>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.13.3</version>
    </dependency>
</dependencies>


  groupId: Name der Organisation (wie ein umgekehrter Paketname).

  artifactId: Name der Bibliothek.

  version: Die exakte Version.
```

Use Case 2: Das Projekt bauen

Maven definiert einen Standard-Lebenszyklus (Build Lifecycle) mit Phasen, die über das Terminal aufgerufen werden.

```
Wichtige Befehle:

    mvn compile: Kompiliert den Quellcode.

    mvn test: Führt alle automatisierten Tests aus.

    mvn package: Führt alle vorherigen Phasen aus und verpackt das Ergebnis in
eine .jar-Datei (im target-Ordner).

    mvn clean: Löscht den target-Ordner für einen sauberen Neubau.
```

Use Case 3: Die Standard-Projektstruktur

Maven erzwingt eine einheitliche Ordnerstruktur, die zum weltweiten Standard geworden ist.

```
src/main/java: Java-Quellcode der Anwendung.

src/main/resources: Konfigurationsdateien, Bilder, etc.

src/test/java: Java-Quellcode der Tests.

target/: Generierte Dateien (.class, .jar).
```

Teil 3: Vertiefung (JavaMasta's Profi-Tipps)

```
Abhängigkeits-Scopes: Mit <scope>test</scope> deklariert man eine Abhängigkeit
(z.B. JUnit), die nur für Tests benötigt und nicht in die finale Anwendung mit
verpackt wird. Das hält die Anwendung schlank.

Maven Wrapper (mvnw): Eine kleine Skript-Datei im Projekt, die es ermöglicht, das
Projekt zu bauen, ohne Maven global installiert zu haben. Der Wrapper lädt bei der
ersten Ausführung automatisch die korrekte, im Projekt definierte Maven-Version
herunter und sorgt so für maximale Konsistenz.

Maven vs. Gradle: Gradle ist ein moderneres Build-Tool, das oft als flexibler und
performanter gilt (es verwendet Groovy/Kotlin-Skripte statt XML). Maven ist der
ältere, sehr weit verbreitete und oft als einfacher empfundene Standard. Das
Verständnis von Maven ist eine essenzielle Fähigkeit für Java-Entwickler.
```