Einführung in *Apache Maven*

- Werkzeug, um ein Projekt aufzusetzen
  →Automatisiert und vereinfacht viele in der Softwareentwicklung anfallende Prozeduren, indem es Standardkonventionen und –praktiken anwendet
  →Entwickler sollen von der Anlage eines Projekts über das Kompilieren, Testen und „Packen“ bis zum Verteilen der Software auf Anwendungsrechner so unterstützt werden, dass viele Schritte automatisiert werden können
- Kostenloses Build-Werkzeug zum standardisierten Erstellen und Verwalten von Java-Programmen (Geschrieben in Java und in erster Linie zum Java-Programm-Builden genutzt)
- Apache Software Foundation
- *Maven* – „Sammler des Wissens“
- „Konvention vor Konfiguration“ (Man braucht sehr wenige Konfigurationseinstellungen hinterlegen, um den Lebenszyklus eines Softwareprojekts abzubilden)
- Builds, Documentation, Reporting, Dependencies , SCMs, Releases, Distribution

Vorteile von Maven:

- ✓ Fördert Standardisierungen, Convention over Configuration und Realisierung von Best Practice
- ✓ Wiederverwendung, einheitliche Verzeichnisstrukturen und Organisation der Abhängigkeiten
- ✓ Vereinfacht das Handling bei vielen Abhängigkeiten und benötigten Zusatz-Artefakten
- ✓ Fördert durch die Definition von Goals in Plug-ins die Arbeitsteilung zwischen Konfigurationsmanagement und Softwareentwicklung
- ✓ Bietet Unterstützung und Anbindung für weitere Anwendungen (Fehlerverfolgung, Reporting-Systeme, Integrationssysteme)

Installation:

- Java-Tool → Java muss installiert sein, Java Software Developer Kit wird benötigt
- Download von Maven
- Umgebungsvariablen an die Java- und Maven-Verzeichnisse anpassen

Projekt anlegen:

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -
DinteractiveMode=false
```

Maven-Archetyp-Mechanismus: `mvn archetype:generate`

Archetyp: Muster/Modell, aus dem alle anderen Dinge der gleichen Art hergestellt werden
→Template eines Projektes

Standard-Verzeichnisstruktur:

```
my-app
|-- pom.xml (Project Oject Model)
`-- src (Alle Eingabedateien)
    |-- main (Eingabedateien für die Erstellung des eigentlichen Produkts)
    |   `-- java (Java-Quelltext)
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    |
    `-- test (Eingabedateien, die für automatisierte Testläufe benötigt werden)
        `-- java (Junit-Testfälle für automatisierte Tests)
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

Die Konfigurationsdatei *POM.xml*:

- *Project Object Model* (Projektmodell)
- Projektbeschreibungs- und steuerungsdatei
- Speichert Informationen für ein Softwareprojekt (Metadaten)
  →Angaben über die verwendete Projektstruktur, Spezifizierung der externen Bibliotheken
- Folgt einem standardisierten Format
- Wird Maven ausgeführt, prüft es zunächst, ob die POM.xml alle nötigen Angaben enthält und ob alle Angaben syntaktisch gültig sind
- Fünf Bereiche: Koordinaten, Projektbeziehungen, Projektinformationen, Projekteinstellungen, Projektumgebung
- Wichtige Koordinaten: *(identifizieren ein Artefakt)*
  - `project`
  - `modelVersion`
  - `groupId`: Eindeutige Kennung der Organisation oder Gruppe des Projekts
    - `artifactId`: Eindeutiger Basisname des primären Artefakts, das generiert wird
      Der vollständige Dateiname des Hauptergebnis-Artefakt wird meist gebildet aus:
      `artifactId>-<version>.<type>`
      `junit-3.8.1.jar`
      `MeineApp-1.0-SNAPSHOT.jar`
      Falls ein `classifier` definiert ist, wird auch dieser noch dem Dateinamen hinzugefügt
      `testng-5.11-jdk15.jar`
  - `version`: In einer „Release"-Version ist der Code unveränderlich
    - SNAPSHOT: zeigt an, dass sich ein Projekt im Entwicklungsstadium befindet, bietet keine Garantie, dass der Code stabil oder unveränderlich ist
      →"Entwicklungs"-Version vor der endgültigen „Release"-Version
  - `name`: Anzeigename für das Projekt
  - `url`: Website des Projekts
  - `properties`: Enthält Wert-Platzhalter, auf die überall innerhalb eines POMs zugegriffen werden kann
  - `dependencies`
    - `scope`: Sichtbarkeit, in welchen Phasen des Buildprozesses die Abhängigkeit benötigt wird
  - `build`: Beinhaltet Deklaration der Verzeichnisstruktur und die Verwaltung von Plug-ins

- Weitere (optionale) Koordinaten:
  - `classifier`: Definieren unterschiedliche Ausführungen der Ergebnis-Artefakte
  - `packaging`: Z.B. jar, war, pom, maven-plugin

Das Projekt bauen:

`mvn package` (validate, generate-sources, process-sources, generate-resources, process-resources, compile)

Das neu kompilierte und verpackte jar testen: `java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App`

Standard-Build-Lebenszyklus:

- Prozess zur Erstellung und Verteilung eines bestimmten Artefakts (Projekts) ist klar definiert
- `clean`: Löschen des Zielverzeichnis, bereinigt Artefakte, die von früheren Builds erzeugt wurden, vorkonfigurierte Plug-ins
- `default` bzw. `build`: Ausführen des Phasen, keine vorkonfigurierten Plug-ins
- `site`: Zusammenstellen von Webseiten, die als Standard-Projektdokumentation dienen, vorkonfigurierte Plug-ins
- Jeder Build-Lebenszyklus wird durch eine Liste von Build-Phasen definiert, wobei eine Phase eine Stufe im Lebenszyklus darstellt

Maven-Phasen des `default`-Zyklus:

- Zyklus wird bei der Softwareerstellung häufig durchlaufen
  →Allerdings muss nicht jedes Softwareprojekt *alle* Phasen verwenden
- Phasen werden in einer bestimmten Reihenfolge durchlaufen
  Wird im Kommanozeilenfenster z.B. das Kommando „mvn package“ eingegeben, dann werden alle vorhergehenden Phasen und die angegebene Phase ausgeführt, aber nicht die nachfolgenden

- `archetype`: Damit kann ein Template für ein Softwareprojekt erstellt werden
- `validate`: Es wird geprüft, ob die Projektstruktur gültig und vollständig ist
- `compile`: Quellcode kompilieren
- `test`: Kompilierter Code wird mit einem passenden Testframework getestet
- `package`: Der kompilierte Code wird ggf. mit anderen nicht-kompilierbaren Dateien zur Weitergabe verpackt, es handelt sich um eine Jar-Datei
- `integration-test`: Softwarepaket wird auf eine Umgebung (anderer Rechner, anderes Verzeichnis, Anwendungsserver) geladen und seine Funktionsfähigkeit geprüft
- `verify`: Prüfungen, ob das Softwarepaket eine gültige Struktur hat und ggf. bestimmte Qualitätskriterien erfüllt
- `install`: Installieren des Softwarepakets in dem lokalen Maven-Repository, um es in anderen Projekten verwenden zu können
- `deploy`: Installieren im fernen Maven-Repo, Versionen stehen damit in Umgebungen mit mehreren Entwicklern allen zur Verfügung (Gemeinsame Nutzung)

In den Lebenszyklus-Phasen werde jeweils bestimmte Plugin-Goals ausgeführt:

Maven-Plugin:

- Bibliotheken, die thematisch zusammengehörenden Goals implementieren

Goal:

- Von Maven-Plugins angebotene Kommandos
- Können bestimmten Lebenszyklus-Phasen zugeordnet werden und werden dann automatisch zum richtigen Zeitpunkt aufgerufen (z.B. compiler:compile)