Maven-Dokumentation für Java und JavaFX

Erstellt für die Diplomarbeit

Autor: [Dein Name]

Datum: 05.08.2025

# Inhaltsverzeichnis

1 1 1. Einleitung

2 2 2. Grundlagen von Maven

2.1 2.1 2.1 Was ist Maven?

2.2 2.2 2.2 Standard-Projektstruktur von Maven

3 3 3. Aufbau der pom.xml

3.1 3.1 3.1 Wichtige Abschnitte der pom.xml

4 4 4. Integration von Maven in IntelliJ IDEA

5 5 5. JavaFX mit Maven einrichten

6 6 6. Beispielprojekt: JavaFX Hello World

7 7 7. Häufige Probleme und Lösungen

8 8 8. Best Practices

9 9 9. Fazit

Dokumentation: Maven-Konfiguration in IntelliJ für Java und JavaFX

# 1 1. Einleitung

Maven ist ein mächtiges Build-Management-Tool für die Java-Entwicklung. In Kombination mit IntelliJ IDEA erleichtert Maven die Verwaltung von Abhängigkeiten, das Bauen von Projekten und die Integration von Frameworks wie JavaFX.  
  
In dieser Dokumentation werden folgende Punkte behandelt:  
1. Grundlagen von Maven  
2. Aufbau der pom.xml  
3. Einrichtung von Maven in IntelliJ  
4. Konfiguration für JavaFX  
5. Beispielprojekte  
6. Häufige Probleme und Best Practices

# 2 2. Grundlagen von Maven

## 2.1 2.1 Was ist Maven?

Maven ist ein Projektmanagement- und Build-Tool für Java, das vor allem die Automatisierung von Build-Prozessen und die Verwaltung von Bibliotheken (Dependencies) vereinfacht.  
  
Hauptmerkmale von Maven:  
- Deklarative Projektkonfiguration via pom.xml  
- Dependency Management  
- Standardisierte Projektstruktur  
- Integration in beliebte IDEs wie IntelliJ IDEA

## 2.2 2.2 Standard-Projektstruktur von Maven

Ein Maven-Projekt folgt einer festgelegten Struktur:  
  
projektname/  
│  
├─ src/  
│ ├─ main/  
│ │ ├─ java/ # Java-Quellcode  
│ │ └─ resources/ # Ressourcen wie FXML, Bilder, etc.  
│ └─ test/  
│ ├─ java/ # Testklassen  
│ └─ resources/  
│  
├─ target/ # Kompilierte Dateien  
└─ pom.xml # Maven-Konfiguration

# 3 3. Aufbau der pom.xml

Die pom.xml ist das Herzstück eines Maven-Projekts. Sie definiert unter anderem:  
- Projektinformationen  
- Dependencies  
- Build-Plugins  
- Java-Version und Properties  
  
Ein Beispiel für eine einfache pom.xml:

<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>com.example</groupId>  
 <artifactId>javafx-demo</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
</project>

## 3.1 3.1 Wichtige Abschnitte der pom.xml

- <dependencies>: Hier werden alle benötigten Bibliotheken aufgelistet  
- <build>: Build-Konfiguration und Plugins  
- <properties>: Globale Projekteinstellungen wie Java-Version  
- <repositories>: Optionale, zusätzliche Repository-Quellen

# 4 4. Integration von Maven in IntelliJ IDEA

Schritte:  
1. Neues Maven-Projekt erstellen (File > New > Project > Maven)  
2. GroupId und ArtifactId festlegen  
3. Java-Version auswählen  
4. IntelliJ generiert automatisch eine Standardstruktur mit pom.xml  
5. Dependencies werden automatisch aus dem Maven Central Repository geladen

# 5 5. JavaFX mit Maven einrichten

JavaFX benötigt eigene Abhängigkeiten, da es seit Java 11 nicht mehr im JDK enthalten ist.  
  
Beispiel-Dependencies für JavaFX in der pom.xml:

<dependencies>  
 <dependency>  
 <groupId>org.openjfx</groupId>  
 <artifactId>javafx-controls</artifactId>  
 <version>20.0.1</version>  
 </dependency>  
 <dependency>  
 <groupId>org.openjfx</groupId>  
 <artifactId>javafx-fxml</artifactId>  
 <version>20.0.1</version>  
 </dependency>  
</dependencies>

# 6 6. Beispielprojekt: JavaFX Hello World

Eine einfache `HelloFX`-Klasse:

import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Label;  
import javafx.stage.Stage;  
  
public class HelloFX extends Application {  
 @Override  
 public void start(Stage stage) {  
 Label label = new Label("Hello, JavaFX with Maven!");  
 Scene scene = new Scene(label, 400, 200);  
 stage.setScene(scene);  
 stage.setTitle("HelloFX");  
 stage.show();  
 }  
 public static void main(String[] args) {  
 launch();  
 }  
}

# 7 7. Häufige Probleme und Lösungen

- \*\*Fehler: JavaFX-Runtime nicht gefunden\*\*  
 → Lösung: VM-Optionen setzen: --module-path Pfad/zur/javafx/lib --add-modules javafx.controls,javafx.fxml  
- \*\*Fehler beim Laden von Ressourcen (FXML)\*\*  
 → Lösung: Ressourcen im Verzeichnis src/main/resources ablegen  
- \*\*Abhängigkeiten werden nicht geladen\*\*  
 → Lösung: Maven -> Reload Project in IntelliJ

# 8 8. Best Practices

- Verwende immer eine aktuelle Java- und Maven-Version  
- Halte die pom.xml übersichtlich und kommentiere wichtige Abschnitte  
- Nutze separate Module für größere Projekte  
- Integriere Unit-Tests unter src/test/java  
- Nutze Plugins wie javafx-maven-plugin für den komfortablen Start

# 9 9. Fazit

Mit Maven und IntelliJ IDEA lässt sich die Entwicklung von Java- und JavaFX-Anwendungen stark vereinfachen. Die automatische Abhängigkeitsverwaltung, standardisierte Projektstrukturen und die einfache Integration in die IDE sorgen für effizientes Arbeiten und sauberen Code.