Programmierung 3

im Wintersemester 2024/25

Praktikumsaufgabe 6

Build-Automatisierung mit Maven

19. November 2024

In diesem Arbeitsblatt verwenden Sie Maven, um ein Java-Projekt effizient zu erstellen, zu verwalten und zu erweitern. Die Aufgaben bauen schrittweise aufeinander auf und bereiten Sie auf den Einsatz von Maven in realen Projekten vor. Passend für Ihr kommendes eigenes Projekt.

In Aufgabe 1 erstellen Sie ein neues Maven-Projekt und passen die Konfiguration an. In Aufgabe 2 migrieren Sie Ihren Java-Code aus dem vorherigen Arbeitsblatt in das Maven-Projekt. In Aufgabe 3 integrieren Sie eine Bibliothek in Ihr Maven-Projekt, die in Ihrem Java-Projekt sehr nützlich sein kann.

Heads up! Implementieren Sie Ihre Lösung in der Konsole anstatt in einer IDE wie IntelliJ IDEA. Dies ist eine wertvolle Lernerfahrung, um die Funktionsweise von Maven zu verstehen und den Umgang mit Maven ohne IDE zu üben. Dies ist wichtig, da Maven spätestens im Zusammenhang mit dem Abschlussthema *Automatisierte Workflows mit Continuous Integration* auch ohne IDE verwendet wird. Mit entsprechenden Vorerfahrungen wird Ihnen die Implementierung des Workflows und vor allem das Debuggen wesentlich leichter fallen.

Arbeiten Sie bei der Lösung der Aufgaben in einem Team von 3 bis 4 Personen zusammen, so dass Sie die Aufgaben kollaborativ lösen und Ihre Ergebnisse untereinander diskutieren können. Lösen Sie dabei die Programmieraufgaben im Pair Programming und bearbeiten Sie die anderen Aufgaben in einer für Sie geeigneten Weise.

Aktuelles Java Development Kit (JDK)

Die Installation eines aktuellen *Java Development Kit (JDK)* ist Voraussetzung für diese Praktikumsaufgabe und wird für Ihr Projekt empfohlen. Empfehlenswert sind die LTS (Long-Term Support) Versionen von Eclipse Temurin, der OpenJDK-Distribution der Community-Organisation Adoptium. Informieren Sie sich auf der Seite Which Version of JDK Should I Use? über die Vor- und Nachteile der verschiedenen JDKs.

Dort finden Sie die Empfehlung:

> "Use Adoptium Eclipse Temurin 21 and ensure that your local version matches the CI[1] and production version."

Stellen Sie sicher, dass Sie mindestens die noch gepflegte LTS Version 17 von Temurin installiert haben. Besser ist es jedoch, die aktuelle LTS Version 21 zu installieren.

Überprüfen Sie mit `java -version` in der Konsole, ob die gewünschte Version korrekt installiert ist.

[1] *CI* steht für *Continuous Integration*.

1 Neues Projekt

Lernziele

- ☐ Sie installieren Maven auf Ihrem Computer.
- ☐ Sie erstellen ein neues Maven-Projekt mit Hilfe des *Maven Quickstart Archetype*.
- ☐ Sie passen die Konfiguration des Projekts in der `pom.xml` an.

Strecklernziele

- ☐ Sie aktualisieren die Abhängigkeiten und Plugins in der `pom.xml` auf die neuesten Versionen.

In dieser Aufgabe installieren Sie Maven, erstellen ein neues Maven-Projekt und passen dessen Konfiguration an.

Anschließend können Sie die Abhängigkeiten und Plugins in der `pom.xml` auf die neuesten Versionen aktualisieren.

Tipp: Nützliche Informationen für den Einstieg in Maven finden Sie auf der Seite Getting Started Guide in der Dokumentation zu Maven. Lesen Sie diese Seite, um einen Überblick über die Funktionalität von Maven zu erhalten.

1.1 Installieren von Maven

Installieren Sie die aktuelle Version von Maven auf Ihrem Computer. Informationen zur Installation und zum Download von Maven finden Sie auf der Maven-Website.

Prüfen Sie mit dem Befehl `mvn -v` auf der Konsole, ob Maven läuft und ob die richtige Version installiert ist. Die Ausgabe der Versionsinformationen sollte auch die entsprechende Java-Version anzeigen, also z. B. LTS Version 17 oder besser 21 von Temurin.

1.2 Erstellen des Projekts

Verwenden Sie den Befehl `mvn archetype:generate` des Archetype Plugins um ein neues Maven-Projekt zu erstellen. Vergessen Sie nicht die neueste Version des Archetypes zu verwenden. Diese wird mit der Option `-DarchetypeVersion` angegeben.

Wichtig: Lesen Sie die Seite Naming Conventions bevor Sie mit der Projekterstellung beginnen.

Spezifizieren Sie den Archetype `maven-archetype-quickstart` und geben Sie die erforderlichen Informationen ein:

`groupId` Identifiziert die Gruppe oder Organisation, die das Projekt verwaltet. Es ist üblich, hier die umgekehrte Domänenadresse der Organisation zu verwenden, um die Eindeutigkeit zu gewährleisten. Im einfachsten Fall haben alle Projekte einer Organisation dieselbe `groupId`.

`artifactId` Spezifiziert den Namen des Projekts. Dies ist normalerweise der Name, unter dem das Projekt bekannt ist, und auch der Name der resultierenden Datei (z. B. die JAR-Datei `maven-introduction-0.1-SNAPSHOT.jar`).

`version` Gibt die Projektversion an. Diese Information wird verwendet, um zwischen verschiedenen Entwicklungsständen des Artefakts zu unterscheiden (z. B. `0.7` oder `2.2.3`). Verwenden Sie eine Snapshot-Version, z. B. `0.1-SNAPSHOT`.

`package` Das angegebene Java-Paket[2] dient als Grundlage für die Ordnerstruktur in Ihrem Projekt. Maven verwendet diese Paketangabe um die entsprechenden Verzeichnisse unter `src/main/java` und `src/test/java` zu erstellen.

Wenn das Projekt erfolgreich erstellt wurde, finden Sie in dem Verzeichnis, in dem Sie den Befehl ausgeführt haben, ein neues Verzeichnis mit dem Namen der `artifactId`.

[2] Ein Package in Java ist eine Gruppierung verwandter Klassen und Schnittstellen, die zusammen in einem Verzeichnis organisiert sind. Dies erleichtert die Verwaltung des Codes, vermeidet Namenskonflikte zwischen Klassen und ermöglicht einen kontrollierten Zugriff durch Zugriffsmodifikatoren. Packages können weitere Subpackages enthalten. Weitere Informationen finden Sie im offiziellen Java Tutorial in der Lektion Packages sowie auf der Seite Guide to Java Packages.

Hinweis: Die beiden Standardverzeichnisse `src/main/java` und `src/test/java` werden automatisch erstellt. Erinnern Sie sich an letzte Woche, als Sie diese Verzeichnisstruktur für ein Java-Projekt manuell erstellt haben? Maven nimmt Ihnen diese Arbeit ab.

1.3 Anpassen der Konfiguration

Passen Sie mindestens die folgenden Konfigurationen in der `pom.xml` an:

`name` Ersetzen Sie den vorgegebenen Namen durch einen aussagekräftigen Namen für Ihr Projekt.

`url` Sie können diese Angabe vorerst entfernen.

`maven.compiler.release` Siehe Setting the `--release` of the Java Compiler in der Dokumentation des Maven Compiler Plugins. Geben Sie die Version der Java-Plattform an, die Sie verwenden möchten. Also `17` für die vorherige LTS Version oder besser `21` für die aktuelle LTS Version.

1.4 Aktualisieren der Abhängigkeiten und Plugins

Mit dem Versions Plugin können Sie überprüfen, ob neuere Versionen von Abhängigkeiten und Plugins in Ihrem Projekt vorhanden sind.

Die vom Archetype Plugin erzeugte Datei `pom.xml` enthält zumindest veraltete Versionen von Plugins und, falls Sie nicht die aktuelle Version angegeben haben, auch eine veraltete Version der Abhängigkeit JUnit. Verwenden Sie das Plugin, um die neuesten Versionen zu finden und zu aktualisieren.

1.4.1 Abhängigkeiten

Mit dem Befehl `mvn versions:display-dependency-updates` können Sie veraltete Abhängigkeiten in Ihrem Projekt identifizieren. Rufen Sie diesen Befehl auf und aktuali-

sieren Sie die Abhängigkeiten in der `pom.xml` auf die neuesten Versionen.

Tipp: Die aktuelle Version von JUnit kann im Abschnitt `<dependencyManagement>` der `pom.xml` eingetragen werden.

1.4.2 Plugins

In ähnlicher Weise können Sie veraltete Plugins in Ihrem Projekt ermitteln. Verwenden Sie den Befehl `mvn versions:display-plugin-updates`, um die Plugins auf die neuesten Versionen zu aktualisieren.

Tipp: Orientieren Sie sich an den Plugin-Versionen, wie sie von Maven 3.6.3 benötigt werden.

2 Migration

Lernziele

- ☐ Sie migrieren bestehenden Java-Code in ein Maven-Projekt.
- ☐ Sie führen die Unit-Tests aus, um sicherzustellen, dass die Migration erfolgreich war.
- ☐ Sie vergleichen das manuelle Kompilieren und Testen von Java-Code mit der Verwendung von Maven.

In dieser Aufgabe migrieren Sie den Produktionscode und Ihre Unit-Tests aus Aufgabe 1 der vorherigen Praktikumsaufgabe in ein Maven-Projekt. Sie können dazu das Projekt aus der vorherigen Aufgabe verwenden oder ein neues Maven-Projekt erstellen.

2.1 Klassen migrieren

Kopieren Sie den Produktionscode, d.h. die Klasse `SimpleCalculator`, sowie Ihre Unit-Tests, d.h. die Klasse `SimpleCalculatorTest`, in das Maven-Projekt.

Hinweis: Die Verzeichnisstruktur des Maven-Projekts sollte Ihnen dabei bekannt vorkommen. Im einfachsten Fall können Sie die Dateien Ihrer Lösung von letzter Woche in die gleichen Unterordner des Maven-Projekts kopieren.

2.2 Tests ausführen

Testen Sie, ob die Klassen korrekt migriert wurden. Verwenden Sie dazu den Befehl `mvn clean test`.

Hinweis: Ihre Ausgabe sollte den folgenden Teil enthalten.

```
> mvn clean test
[INFO] Scanning for projects...
[...]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running SimpleCalculatorTest
```

```
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.021 s -- in
↪  SimpleCalculatorTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[...]
```

2.3 Manuelles Kompilieren und Testen im Vergleich zu Maven

Vergleichen Sie die Schritte, die Sie im letzten Arbeitsblatt durchgeführt haben, um das Java-Projekt auf Ihrem Computer einzurichten und die Klasse `SimpleCalculator` manuell zu kompilieren und zu testen, mit den Schritten, die Sie in dieser Aufgabe mit Maven durchführen.

Diskutieren Sie in Ihrem Team: Welche Vor- und Nachteile sehen Sie bei der Verwendung von Maven im Vergleich zur manuellen Vorgehensweise? Überlegen Sie dabei, wie sich diese Unterschiede auf Ihre tägliche Entwicklungsarbeit auswirken könnten.

Verwenden Sie die folgenden Fragen als Denkanstöße:

- Welche Unterschiede gibt es bei der Verwaltung von Abhängigkeiten (z. B. JUnit)?
- Wie hat Maven die Anzahl der Befehle oder die Komplexität Ihrer Arbeit verändert?
- Inwiefern könnte Maven Vorteile in größeren Projekten bieten, z. B. bei Projekten mit vielen Bibliotheken oder mehreren Entwickler:innen?

3 Bibliotheken

Lernziele

- ☐ Sie beschaffen sich die notwendigen Informationen, um eine Bibliothek in ein Maven-Projekt zu integrieren.
- ☐ Sie fügen eine Bibliothek als Abhängigkeit in ein Maven-Projekt ein.
- ☐ Sie nutzen die Funktionalität einer mit Maven eingebundenen Bibliothek in Ihrem Java-Projekt.

Strecklernziele

- ☐ Sie lernen Funktionen von Guava kennen, die Sie bei der Entwicklung von Java-Anwendungen unterstützen.

In dieser Aufgabe fügen Sie Ihrem Maven-Projekt eine Bibliothek hinzu, die Sie in Ihrem Java-Projekt verwenden können.

3.1 Guava: Google Core Libraries for Java

Bibliotheken in der Softwareentwicklung dienen grundsätzlich dazu, die Effizienz und Effektivität der Programmierung zu steigern. Sie vereinfachen und standardisieren wiederkehrende Aufgaben, was Entwickler:innen hilft, Zeit zu sparen und die Fehleranfälligkeit zu verringern. Dies fördert die Erstellung zuverlässiger und wartungsfreundlicher Softwareanwendungen.

Guava stellt eine umfangreiche Sammlung von Hilfsklassen und -methoden zur Verfügung. Diese beinhalten Funktionen für die Arbeit mit Collections, Caching, Primitives, Concurrent Programming, Stringbehandlung und vieles mehr. Guava legt großen Wert auf Performance und Qualität, was durch sorgfältige Optimierung und ausgiebige Tests der Komponenten sichergestellt wird.

Die von Google entwickelte Java-Bibliothek stellt Java-Programmierer:innen eine Vielzahl von Kernfunktionalitäten zur Verfügung. Sie erleichtert und beschleunigt die tägliche Arbeit mit Java durch erweiterte Werkzeuge und Klassen, die über die Standardbibliothek von Java hinausgehen.

Nicht zuletzt aus diesen Gründen gilt Guava als eine der am weitesten verbreiteten Java-Bibliotheken. Um die Lernziele dieser Aufgabe zu erreichen, werden Sie daher diese Bibliothek verwenden.

3.2 Einbinden der Bibliothek

Binden Sie die aktuelle Version von Guava als Abhängigkeit (`<dependency>`) in Ihr Projekt ein. Informationen dazu finden Sie auf der Projektwebsite auf GitHub.

3.3 Verwenden der Bibliothek

Verwenden Sie die `Joiner`-Klasse[3] von Guava, um die an die `main`-Methode übergebenen Argumente zu einem String zu verknüpfen und diesen in die Ausgabe von "Hello World" zu integrieren.

Die Ausgabe könnte z. B. so aussehen:

```
Hello Emilia, Noah, Mia, Luca!
```

Hinweis: Rufen Sie Ihre Implementierung mit dem Befehl `java -cp` auf und geben Sie den passenden Klassenpfad an. Die JAR-Datei Ihres Projekts finden Sie im Verzeichnis `target`. Die von Guava benötigte JAR-Datei befindet sich im lokalen Maven-Repository. Alternativ können Sie herausfinden, wie Sie ein Java-Programm mit dem Exec Maven Plugin ausführen können.

3.4 Erkunden der Bibliothek

Machen Sie sich mit den Funktionen von Guava vertraut, indem Sie das Benutzerhandbuch auf der Projektwebsite lesen. Experimentieren Sie mit verschiedenen Klassen und Methoden, um ein Gefühl für die Möglichkeiten von Guava zu bekommen.

[3] Siehe String utilities / Joiner im Wiki von Guava auf GitHub.

Beispielsweise ist die Methode `Preconditions.checkArgument` nützlich, um die Gültigkeit von Argumenten als Vorbedingung zu überprüfen:

```
checkArgument(args.length > 0, "Please provide some names!")
```