HSLU Hochschule Luzern

Anleitung:
Professionelle Entwicklung von Java-Projekten

Themen: Java-Projekte, Template-Projekt, Unit-Testing, Versionskontrolle mit Git und GitLab, Buildautomatisation, Codequalität, Entwicklungsumgebung, Konfiguration etc.

Roland Gisler, Version 4.1.2 (HS 2022)

Inhalt

1. Einleitung

Dieses Dokument beschreibt und dokumentiert als Zusammenfassung in einer logisch aufbauenden, zunehmenden Detaillierung die wesentlichen Schritte, um von ersten einfachen Java-Projekten bis hin zur professionellen Entwicklung ganzer Applikationen in einem Team zu gelangen.

Das umfasst neben den fundamentalen Installationen von Java, Buildwerkzeugen und der Entwicklungsumgebung auch den Aufbau des auf einem Template basierenden Projektes (für Java-Projekte), den darin integrierten Buildprozess (Apache Maven), den Einsatz von verschiedenen Technologien zur Testautomatisierung, die Arbeit mit dem Versionskontrollsystem (z.B. git / GitLab), sowie die Nutzung von Buildserver- und QS-Diensten mit dem Ziel der Entwicklung nach den Prinzipen der Continuous Integration (CI).

Die Anleitung richtet sich somit mit Schwerpunkt auf das Tooling der HSLU Informatik (NetBeans, GitLab, Jenkins etc.) und ist eng mit den Modulen der Softwareentwicklung (OOP/PRG/PLAB, AD, SWDE, VSK und SWDA) verknüpft, enthält aber auch Hinweise auf Methoden und Werkzeuge, die darüber hinausgehen.

Teil 1 – Einfache Java Projekte

Grundlagen

Module OOP, PLAB, PRG und AD

2. Java: Installation Java Development Kit (JDK)

Die Basis für die Entwicklung von Java-Projekten ist das **J**ava **D**evelopment **K**it (JDK) und das **J**ava **R**untime **E**nvironment (JRE). Während das JRE die Laufzeitumgebung für Java-Projekte (die **J**ava **V**irtual **M**achine, JVM) zur Verfügung stellt, sind im JDK auch die Entwicklungswerkzeuge (wie z.B. der Kompiler) enthalten.

Hinweis: Alle Angaben beziehen sich vorwiegend auf Windows-Systeme.

Voraussetzungen

- Sie benötigen das aktuelle Installationspaket für das Java Development Kit (**JDK 17 LTS**[1]) und dessen Dokumentation. Beziehen Sie dieses entweder direkt von Oracle http://www.oracle.com/technetwork/java/javase/downloads/index.html) oder aus dem vom Dozenten zur Verfügung gestellten SW-Archiv auf SWITCHdrive: http://bit.ly/2OH3Uhh.

Ziele

- Java 17 lauffähig auf Ihrem System installiert.

Installation

Wir müssen nur das JDK herunterladen und installieren, da dieses bereits ein JRE beinhaltet (die JRE-Installation wird automatisch und integriert gestartet). Oracle schlägt bei der Installation Pfäde vor. Es wird empfohlen, diese entsprechend der folgenden Anleitung zu ändern.

1.) Starten Sie die Installation und akzeptieren Sie die Lizenzbedingungen.

2.) Als Verzeichnis für das JDK empfehlen wird einen Namen in der Art **`…/jdk-17.0.4`** zu verwenden (in Abweichung zum Default). Es ist zwar etwas unkonventionell, aber installieren Sie das JDK doch einfach in den folgenden Pfad: **`C:\jdk-17.0.4`** (gilt natürlich nur für Windows; Linux und Unix haben bereits vernünftige, kurze Verzeichnispfäde **ohne** Leerzeichen und Sonderzeichen.)

3.) Nach Abschluss der Installation erstellen Sie eine Umgebungsvariable (Environmentvariable) mit dem Namen **`JAVA_HOME`**, welche auf das Basis-Verzeichnis Ihrer JDK-Installation zeigt (Pfad entsprechend wie in Schritt 2 gewählt anpassen):

```
C:\jdk-17.0.4
```

Erklärung: Es gibt Java-Anwendungen, die auf diese Environment-Variable achten, damit sie wissen mit welcher Java-Version sie ausgeführt werden sollen. Beachten Sie, dass Sie diese Umgebungsvariable bei der Installation einer neuen Version von Java ggf. anpassen müssen.

4.) Ergänzen Sie den Suchpfad (**`Path`**-Umgebungsvariable) Ihres Systems mit dem Eintrag

```
C:\jdk-17.0.4\bin;<restlicher Pfad>
```

damit die Binaries von Java in der Shell[2] gefunden werden. Achten Sie darauf, dass es sich um den **ersten** Eintrag handelt (ganz links bzw. ganz oben).

6.) Entpacken Sie die Zip-Datei mit der kompletten Dokumentation von Java (z.B. **`jdk-17.0.4_doc-all.zip`**) in das Basisverzeichnis des JDK. Da die Zip-Datei ein **`docs`**-Verzeichnis enthält, steht Ihnen die Dokumentation danach auch offline (ohne Internetzugang, schneller) unter **`C:\jdk-17.0.4\docs\index.html`** zur Verfügung.

Tipp: Erstellen Sie eine Bookmark/Shortcut auf die Dokumentation!

---

[1] LTS steht für «Long-Term Support», d.h. eine Java-Version welche für zwei bis drei Jahre Updates erhält.

[2] Shell: Eingabeauforderung, Kommandozeile, Terminal, bash etc.

Verifikation / Test

Öffnen Sie eine Shell und testen Sie ob die Installation erfolgreich war. Starten Sie dazu die JVM mit dem folgenden Befehl:

```
$ java -version
```

Sie sollten eine ähnliche Ausgabe wie folgt erhalten, die exakten Versionsinformationen können natürlich abweichen, wichtig ist die **17.0.x**-Version):

```
$ java -version
java version "17.0.4.1" 2022-08-18 LTS
Java(TM) SE Runtime Environment (build 17.0.4.1+1-LTS-2)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.4.1+1-LTS-2, …xed mode, sharing)
```

Damit ist die Java Virtual Machine des Java Runtime Environments lauffähig. Starten Sie danach probehalber auch noch den Java-Kompiler mit folgendem Befehl:

```
$ javac -version
```

Auch hier könnte die exakte Versionsinformationen in der dritten Stelle abweichen (17.0.**x**):

```
$ javac -version
javac 17.0.4.1
```

Damit haben Sie die notwendigen Voraussetzungen, um mit der Installation weiterer Komponenten fortzufahren. Im Fehlerfall prüfen Sie kritisch nochmal alle einzelnen Schritte.

Wichtig: Unter Windows müssen Sie nach Änderungen von Umgebungsvariablen die Shell jeweils neu starten!

3. Entwicklungsumgebung (IDE): Installation und Konfiguration

Unabhängig davon welche IDE[3] Sie verwenden, sollten Sie in dieser noch einige fundamentale Konfigurationen vornehmen um die Effizienz bei der Entwicklung zu steigern.

Voraussetzungen

- Java Development Kit und Java Runtime Environment lauffähig installiert.

Ziele

- Installation der IDE (z.B. NetBeans IDE, Eclipse JDT, IntelliJ IDEA etc.)
- Wichtige IDE-Grundkonfigurationen als Basis für professionelle Entwicklung.
- Vermeidung von Problemen bei Austausch/gemeinsamer Bearbeitung von Code.

Grundlagen

Beachten Sie: Hier werden nur NetBeans IDE und Eclipse JDT berücksichtig. Die Einstellungen werden aber sinngemäss für alle IDEs empfohlen!

Vorgehen

1.) Laden Sie sich das Installationspaket Ihrer gewünschten IDE herunter (die aktuellen Versionen stehen Ihnen auch auf dem SWITCHdrive http://bit.ly/2OH3Uhh zur Verfügung):
   - NetBeans IDE: https://netbeans.apache.org/download/index.html (Java SE)
   - Eclipse JDT: https://www.eclipse.org/downloads/packages/ (for Java Developer)
   - IntelliJ IDEA: https://www.jetbrains.com/idea/download/ (Community)
   - Visual Studio Code: https://code.visualstudio.com/#alt-downloads

   Für die Mehrheit all unserer Projekte (auch in den Folgemodulen!) reichen die **kleinsten** Editionen vollständig aus. Sie enthalten weniger Ballast, sind übersichtlicher und schneller. Und wenn dennoch etwas fehlen sollte, lässt sich das später problemlos gezielt ergänzen.

2.) Installieren Sie Ihre gewählte Entwicklungsumgebung mit dem Installer und prüfen Sie danach ob sie läuft. Wenn Sie **keine** Präferenz und **wenig/keine Erfahrung** mit IDE's haben, empfehlen wir Ihnen die **NetBeans IDE**. Sie ist einfach und leicht verständlich.

3.) Suchen Sie in Ihrer IDE nach den entsprechenden Orten, um die folgenden Konfigurationen vorzunehmen (nächster Abschnitt, Hinweise jeweils für NetBeans und Eclipse enthalten).

   Hinweis: Die Konfigurationen sind bei manchen IDE's global, bei anderen jeweils «nur» auf einen konkreten **Workspace** (Vereinfacht erklärt: Verzeichnis, in welchem sich mehrere Projekte befinden) oder sogar nur auf das jeweilige Projekt wirksam bzw. festlegbar!

Konfiguration

1.) **Java-Plattform**: Legen Sie das korrekte (Default-)**JDK** fest.

   Dazu wird in der Regel das **`JAVA_HOME`**-Verzeichnis und die Java-Version (z.B. **`17.0.4`**) konfiguriert. Bei der Erstinstallation wird das korrekt sein, aber wenn Sie später eine neuere Java-Versionen installieren ist das immer zu überprüfen. Es ist durchaus möglich das verschiedene Projekte auch mit verschiedenen Java-Versionen arbeiten, das kann z.B. bei Migrationen sogar sehr nützlich sein. Haben Sie wie empfohlen auch die Java-Dokumentation installiert, ergänzen Sie in der Konfiguration auch deren Pfad (**`$JAVA_HOME/docs/api`**).

   NetBeans: **`Tools → Java Plattforms`**

   Eclipse: **`Window → Preferences`**, Suchen nach: «**`Installed JREs`**»

[3] IDE = **I**ntegrated **D**evelopment **E**nvironment, sinngemäss «Integrierte Entwicklungsumgebung»

2.) **Encoding**: Setzten Sie das **Encoding** aller Dateien auf **`UTF-8`**.

Das Encoding beeinflusst u.a. die Codierung von Sonderzeichen (wie zum Beispiel «ä», «ö» und «ü»). Die meisten IDE's verwenden per Default das «Standard-Encoding» des Betriebssystems. Das ist leider nur bedingt eine gute Idee: Windows verwendet z.B. **`cp1252`** als Default-Encoding, viele Linux/Unit hingegen **`latin-1/UTF-8`**. Schreibt also ein Kollege z.B. auf Windows einen schönen JavaDoc-Kommentar, versteht die Kollegin auf dem Ubuntu mindestens die Umlaute nicht mehr!

NetBeans: Individuell pro Projekt, im Template-Projekt automatisch bereits festgelegt.

Eclipse: **`Window→Preferences,`** Suche «**`Workspace`**», siehe «**`Text file encoding`**»

3.) **Codeformatierung**: Einrückung mit **vier Leerzeichen**, Zeilenlänge auf **120 Zeichen**.

In der Regel basieren die IDE's automatisch auf den Java Code Conventions. Doch gibt es auch da noch kleinere Freiheitsgrade. In der Praxis hat es sich bewährt, für die Einrückung von Code Leerzeichen (statt Tabulatoren) zu verwenden, da die meisten Texteditoren Tabs von acht (oder mehr) Zeichen definieren.
Über die Einrücktiefe (in der Regel mindestens zwei bis vier Zeichen) als auch die Zeilenlänge (80 oder mehr Zeichen) kann man ebenfalls trefflich streiten. Tatsächlich sind die konkreten Werte weniger wichtig, als dass man sich in einem Team auf **einheitliche** Werte einigt. Ansonsten wird der Code von jedem Mitglied wieder anders formatiert. Das ist spätestens im Zusammenhang mit Versionskontrollsystemen sehr schlecht, da es immer «Änderungen» produziert, die gar keine sind.

NetBeans: **`Tools→Options→Editor→Formatting`**

Eclipse: **`Window→Preferences; Java→Code Style→Formatter`**

4.) **Code-Templates**: Individuelle Vorgaben, z.B. für Lizenz oder JavaDoc.

Diese Konfiguration kann zwar aufwändig sein, erleichtert die Arbeit aber ungemein: Die meisten IDE's erlauben es Ihnen individuelle Templates für neue Klassen, neue Methoden, Setter/Getter, JavaDoc-Kommentare usw. zu definieren. Was sich im Minimum lohnt ist zumindest das Entfernen von Dingen, die Sie immer als erstes wieder löschen (z.B. den Default-Kommentar «write your documentation here» o.ä.). Auch mit der Definition von Codeskeletten für automatisierte Tests lässt sich sehr viel Zeit sparen!

NetBeans: **`Tools→Options→Editor→Code Templates`**

Eclipse: **`Window→Preferences; Java→Code Style→Code Templates`**

5.) **Buildwerkzeug**: Konfiguration des **Buildtools**.

Sofern Sie eine individuelle Installation eines Buildtools (z.B. Apache Maven) vorgenommen haben (optional, aber sehr empfohlen, siehe Kapitel 4), lohnt es sich sehr, die z.T. bereits in den IDE's enthaltenen Tools (internal) durch die externen Tools zu ersetzten. Das hat den positiven Effekt, dass der Buildprozess nachvollziehbar **immer** und in jeder Umgebung **identisch** ausgeführt wird. Damit erreichen Sie (fast) eine «single point of configuration» Semantik.

NetBeans: **`Tools→Options→Tab Java→Maven`**

Eclipse: **`Window→Preferences; Maven→Installations`**

6.) **NetBeans**: Diverse Konfigurationen in NetBeans

Die folgenden, NetBeans-spezifischen Konfigurationen machen die Programmierung einfacher und leichter (alles unter **`Tools → Options`**):

- Unter **`Editor→Hints`** für Java alle vorhanden Hints aktivieren (auswählen)
- Unter **`Editor→On Save`** für Java alle Optionen anwählen.

4. Build: Installation und Konfiguration Apache Maven

Apache Maven ist ein Werkzeug, dass Sie beim «Bauen» (build, Buildprozess) von Softwareprojekten unterstützt. Damit sind die immer wiederkehrenden Tätigkeiten, wie z.B. Kompilieren, JAR-Dateien erzeugen, Tests ausführen, Javadoc erzeugen etc. gemeint. All das kann mit Apache Maven automatisiert werden. Der Vorteil liegt auf der Hand: Der Buildprozess ist damit nicht nur schneller, sicherer und einfacher, sondern er wird damit auch (unabhängig von einer Entwicklungsumgebung und der Plattform!) jederzeit und überall reproduzierbar.

Voraussetzungen

- Java Development Kit und Java Runtime Environment lauffähig installiert.

Ziele

- Installation von Apache Maven
- Basiskonfiguration für den Einsatz in der Shell.
- Optional: Integration in die IDE (empfohlen).

Installation

1.) Laden Sie die **binäre** (binary archive) Distribution von Apache Maven herunter. Sie finden sie auf dem SWITCHdrive http://bit.ly/2OH3Uhh oder von der Originalquelle unter https://maven.apache.org/download.cgi.

2.) Die Installation verfügt über kein Setup, sondern besteht aus einem einfachen Entpacken der Zip-Datei. Wir empfehlen auch hier einen kurzen Pfad, wie z.B. **`C:\apache-maven-3.8.6`** zu verwenden.

3.) Ergänzen Sie den Suchpfad **`Path`** (Umgebungsvariable) mit dem **`\bin`**-Verzeichnis der Maven-Installation:

```
C:\apache-maven-3.8.6\bin;<restlicher Pfad>
```

Damit können Sie das Maven-Binary `mvn` in einer Shell von einem beliebigen Verzeichnis aus aufrufen.

Verifikation / Test

Testen Sie die Installation, indem Sie eine Shell öffnen und

```
$ mvn -version
```

eingeben. Sie sollten die Version der installierten Maven- und Java-Versionen sehen:

```
$ mvn -version
Apache Maven 3.8.6 (84538c9988a25aec085021c365c560670ad80f63)
Maven home: C:\apache-maven-3.8.6
Java version: 17.0.4.1, vendor: Oracle Corporation, runtime: C:\jdk-17.0.4
Default locale: de_CH, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Erweiterte Konfiguration

1.) Rufen Sie Maven **ohne** Argumente auf:

```
$ mvn
```

Sie erhalten darauf eine Fehlermeldung die darauf hindeutet, dass sich im aktuellen Verzeichnis **kein** Maven-Projekt befindet. Mit diesem Aufruf wurde aber in Ihrem **HOME**-Verzeichnis ein **.m2**-Verzeichnis erstellt (z.B. unter Windows: **C:\Users*<uid>*\.m2**). Überprüfen Sie das! Wenn es nicht existiert, legen Sie das Verzeichnis selber an!

2.) Legen Sie im oben erwähnten **.m2**-Verzeichnis eine **settings.xml**-Datei mit folgendem Inhalt an[4] (Hinweis: Die Datei ist auch im Projekttemplate (siehe nächstes Kapitel) im Verzeichnis **.mvn** abgelegt):

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
   http://maven.apache.org/xsd/settings-1.0.0.xsd">
   <mirrors>
      <mirror>
         <id>hslu-nexus</id>
         <name>HSLU Nexus Repository</name>
         <url>https://repohub.enterpriselab.ch/repository/maven-public/</url>
         <mirrorOf>*</mirrorOf>
      </mirror>
   </mirrors>
</settings>
```

Hintergrund: Maven unterstützt ein so genanntes Dependency-Mangement, welches auf einem binären Repository (Server) basiert. Mit dieser Konfigurationsdatei wird ein HSLU-internes Repository (als Mirror) verwendet, was die Netzbelastung senkt, die Dependency-Auflösung beschleunigt und uns später die Möglichkeit gibt auch eigene Dependencys dort zu deployen und aufzulösen.

3.) Ergänzen/korrigieren Sie die Konfiguration Ihrer IDE mit der soeben installierten Maven-Version, damit die Entwicklungsumgebung nicht mehr die interne (bundled), sondern die soeben installierte (external) Version verwendet.

NetBeans IDE: **Tools → Options → Tab «Java» → Maven**

Eclipse JDT: **Windows → Preferences; Maven → Installations**

IntelliJ IDEA: **File → Settings → search «Maven» → Maven Home Dir**

[4] Apple-UserInnen empfehlen wir für die Bearbeitung von Text-Dateien «TextWrangler».

5. Projekttemplate: Grundlagen und Verwendung

Zum einfacheren Einstieg in die Entwicklung stellen wir Ihnen ein «leeres» Java-Projekt «Template» zur Verfügung, welches ein weitgehend vorkonfiguriertes Java-Projekt mit automatisiertem Buildprozess enthält, so dass Sie auf dessen Basis neue Projekte erstellen können. Es enthält unter anderem eine standardisierte Verzeichnisstruktur, die im Kapitel 6 erklärt wird. **Wichtig**: Alle in diesem Kapitel nicht speziell vermerkten Anleitungen zur IDE beziehen sich auf die NetBeans IDE! Hinweise zu anderen IDE's erhalten Sie in den spezifischen Abschnitten am Schluss dieses Kapitels.

Voraussetzungen

- «OOP Maven Template» (wird als ZIP-Datei zur Verfügung gestellt).
- Eine installierte IDE (z.B. NetBeans IDE), siehe Kapitel 3.
- Optional: Apache Maven installiert und in Shell (und IDE) lauffähig, siehe Kapitel 4.

Ziele

- Einrichten eines auf dem HSLU Java Template basierenden Projektes.

Grundlagen

Das «HSLU Java Template» wird Ihnen als ZIP-Datei (**`oop_maven_template_jdk17-3.2.0.zip`**) auf dem ILIAS oder auf SWITCHdrive http://bit.ly/2OH3Uhh (im Verzeichnis **`15_maven_template`**) zur Verfügung gestellt. Das darin vorbereitete Projekt integriert sich, da es auf dem Apache Maven Buildsystem basiert, bei den gängigen IDE's (fast) vollautomatisch.

Einrichten eines neuen Projektes

Wenn Sie ein neues Projekt erstellen wollen, entpacken Sie die ZIP-Datei des Templateprojektes in Ihr gewünschtes Verzeichnis[5], in welchem Sie das Projekt ablegen bzw. bearbeiten wollen. Das Projekt (und das Verzeichnis) hat dann den Namen «**`oop_maven_template`**», welchen Sie natürlich anpassen sollen!
Am einfachsten funktioniert das mit der NetBeans IDE: Öffenen Sie das Projekt über **`File → Open Project…`** und wählen Sie das entpackte Verzeichnis aus. Nachdem das Projekt geöffnet ist, rufen Sie im Kontextmenu des Projektes die Funktion «**`Rename`**» auf und markieren/ändern alle drei Namen einheitlich auf den von Ihnen gewünschten Namen (Beispiel: **`oop_exercises`**)

Wenn Sie das Projekt nur in der Shell oder in einer anderen IDE bearbeiten wollen, gehen Sie wie folgt vor:

1. Benennen Sie das Verzeichnis **`oop_maven_template`** auf den gewünschten Namen um.
2. Öffnen Sie die darin enthaltene Datei **`pom.xml`**, und tragen Sie in den XML-Elementen **`artifactId`** (Zeile **22**) und **`name`** (Zeile **26**) denselben Namen ein.

Nun können Sie das Projekt in einer beliebigen IDE öffnen (bzw. bei Eclipse müssen Sie es einmalig **importieren**) und die Namensgebung ist bereits korrekt.

Ausführen von einfachen Standard-Build-Goals

Die fundamentalen Build-Goals können Sie ungeachtet des Templates ganz normal und direkt in der IDE über das (Kontext-)Menu des Projektes ausführen. Der einzige Unterschied ist, dass im Output-Bereich die entsprechenden Konsolenausgaben von Apache Maven sichtbar sind.

Drücken Sie in NetBeans einfach mal die Taste «F11» und schauen Sie was passiert!

[5] Wählen Sie ein kurzes, einfaches Basisverzeichnis, ohne jede Sonderzeichen in der Namensgebung!

Ausführen von spezifischen Build-Goals

Sofern Sie sich mit Apache Maven bereits etwas auskennen (oder sich näher damit auseinandersetzen wollen) können Sie über das Kontextmenu des Projektes mit **`Run Maven → Goals…`** jedes beliebige Maven-Goal ausführen, das Sie auch in der Shell ausführen können.

Wichtig: Wenn Sie noch **nie** mit Maven gearbeitet haben, starten Sie als erstes Goal «**`site`**»: Entweder wie oben beschrieben in der Entwicklungsumgebung, oder direkt in der Shell mit:

```
$ mvn site
```

Beachten Sie, dass Sie dafür im Projektverzeichnis sein müssen, also dort wo sich die Datei **`pom.xml`** befindet! Erschrecken Sie nicht, bei der **allerersten** Ausführung wird dieser Prozess einiges aus dem Internet herunterladen, dabei Ihr lokales Repository (**`$HOME/.m2/repository`**) befüllen und entsprechend lange dauern. Danach ist auch alles für die IDE bereit (alle Dependencies werden lokal in einen Cache geladen).
Nebenbei: Öffnen Sie danach mit einem Browser die Datei **`./target/site/index.html`** und staunen Sie!

Verwenden von Thirdparty-Libraries (Abhängigkeiten, Dependencies)

Das Template ist so vorbereitet, dass die wichtigsten Thirdparty-Dependencies für die Module OOP und AD bereits erfasst sind. Diese werden von Apache Maven (welches über ein integriertes Dependency-Management verfügt) automatisch von sogenannten Maven-Repositories[6] heruntergeladen und ebenfalls automatisch in die IDE (u.a. in den Classpath) integriert.

Bei NetBeans sind die Dependencies sauber aufgeteilt unter «Dependencies» und «Test-Dependencies» in der «Projekts-View» sichtbar. Dort können Sie auch gezielt die Quellen (wenn verfügbar) und die JavaDoc herunterladen und betrachten.

Sollten Sie weitere Dependencies benötigen, können Sie diese dort auch ergänzen. Scheuen Sie nicht Ihren Dozenten um Rat zu fragen. Es ist nicht ganz so einfach wie es scheint, da sich der Autor des Templates entschieden hat mit so genannten «Managed Dependencies» zu arbeiten. Diese haben speziell im Hinblick auf spätere und grössere Projekte den Vorteil, dass einheitliche Versionen über das ganze Projekt verwendet werden, sind aber leicht aufwändiger in der Definition.

Ein Blick hinter den Vorhang: `pom.xml`

Sie fragen sich vielleicht woher die IDE denn nun so genau weiss wie und wo alles zu tun ist. Das (offene) Geheimnis dahinter stellt die Datei **`pom.xml`** dar, welche Sie im Basisverzeichnis Ihres Projektes finden. Darin ist alles Wissenswerte zum Projekt deklariert. Werfen Sie mal einen Blick in die Datei. Es gibt darin durchaus einige Elemente die Sie leicht wiedererkennen werden (z.B. im Vergleich mit der generierten Website).

Keine Angst, Sie können diese Datei vorerst getrost als eine «blackbox» betrachten. Wir werden uns erst in späteren Modulen (z.B. SWDE oder VSK) detaillierter damit auseinandersetzen.

Wenn Sie bereits interessiert sind, finden Sie auf der Website http://maven.apache.org/ unter «Documentation» mehr Informationen. Sollten Sie sich wundern, dass die 08/15-Beispiele in den Maven-Tutorials kaum mehr als zehn Zeilen enthalten, unser **`pom.xml`** hingegen mehr als 700 Zeilen umfasst: Das ist die Differenz zur professionellen, reproduzierbaren und nachvollziehbaren Entwicklung. ☺

[6] Maven-Repositories sind Binär-Repositories (also keine SCMs!), welche in wohldefinierter Namensstruktur vorwiegend Libraries (JAR, WAR, EAR's etc.) zum Download anbieten. Entstanden mit Apache Maven (ein Buildwerkzeug), sind sie mittlerweile zu einem Quasistandard geworden und werden von einer vielzahl Werkzeugen unterstützt. Siehe dazu auch http://search.maven.org/

Arbeiten in der Shell (optional)

Sofern Sie die (optionale) Installation von Kapitel 4 vollzogen haben, können Sie alles was wir bisher in der IDE ausgeführt haben auch direkt in der Shell ausführen!

Was im ersten Moment vielleicht etwas spartanisch wirkt, hat in Wirklichkeit sehr viele Vorteile: Es stellt einen wichtigen Grundpfeiler der Automatisation dar, welche für Continuous Integration (CI) notwendig ist (wird im Modul VSK behandelt). Sie können völlig unabhängig von einer IDE, jederzeit und überall, mit minimalen Ressourcen Ihr ganzes Projekt «bauen».

Probieren Sie in der Shell (mit dem gewünschten Projektverzeichnis als Arbeitsverzeichnis) doch mal folgende Kommandos aus (jedes davon startet Maven mit einem «Goal»):

- **`mvn clean`** – räumt erst mal auf und löscht alles, was generiert werden kann.
- **`mvn compile`** – Kompiliert den gesammten Quellcode.
- **`mvn test`** – führt alle (Unit-)Tests aus.
- **`mvn javadoc:javadoc`** – erstellt die Javadoc
- **`mvn install`** – führt alle notwendigen Aufgaben aus, damit das Projekt in binärer Form in Ihr lokales Repository deployed werden kann (beachten Sie die letzten paar Zeilen der Ausgabe oberhalb von "**`BUILD SUCCESS`**"

Und natürlich können Sie auch **`mvn site`** ausführen, was Sie aber bereits früher in diesem Kapitel gemacht haben sollten. Wenn nicht: Holen Sie das nach! Sie erhalten dann eine komplette Website zu Ihrem Projekt, welche im Verzeichnis **`./target/site`** zu finden ist.

Hinweise zu Eclipse

Bei Eclipse **importieren** Sie das Projekt einmalig in Ihren Workspace. Rufen Sie dazu nachdem (!) Sie Ihr Projekt bereits umbenannt haben den Import-Dialog über **`File→Import…`** auf.

In der Liste selektieren Sie den Eintrag **`Maven→Existing Maven Projects`** und gehen mit **`Next`** weiter. Dort wählen Sie über **`Browse…`** als Root-Directory das Projektverzeichnis aus und schliessen den Import mit **`Finish`** ab.

Nach dem erfolgreichen Import finden Sie im Projekt zwei neue Dateien: **`.project`** und **`.classpath`**. Das sind die Eclipse Projektdateien, welche Sie nicht löschen sollten. Sind sie vorhanden, können Sie einmal importierte Projekte beliebig öffnen und schliessen.

Hinweise zu IntelliJ IDEA

Bei IntelliJ können Sie das Projekt direkt öffnen. Wählen Sie dazu im Welcome-Dialog **`Open`** und suchen Sie dann das gewünsche Projektverzeichnis aus. Das Projekt wird geöffnet und auf Basis des **`pom.xml`** korrekt konfiguriert.

Hinweise zu Visual Studio Code

Bei Visual Studio Code können Sie das Projekt direkt öffnen. Das Projekt wird auf Basis des **`pom.xml`** korrekt konfiguriert. Voraussetzung ist aber, dass Sie das Java-Tooling nachinstalliert haben.

6. Java: Einheitliche Projektverzeichnisstruktur

Wir verwenden in allen (Java-)Projekten eine einheitliche Verzeichnisstruktur. Das erlaubt uns nicht nur, uns in verschiedenen Projekten viel schneller zurecht zu finden, sondern stellt auch eine wichtige Basis für die Automatisierung von diversen Verarbeitungen dar.

Voraussetzungen

- Grundkenntnisse zu Java-Projekten (Quellcode, Ressourcen, Konfiguration).

Ziele

- Einheitliche Struktur zur schnellen Orientierung.
- Einfache Adaption von (automatisierten) Abläufen.
- Einhalten von Standards (Apache Maven)

Grundlagen

Die hier vorgestellte Verzeichnisstruktur ist keine Erfindung der HSLU, sondern basiert auf einem Quasi-Standard, welcher von Apache Maven eingeführt wurde, und von zahlreichen anderen Tools ebenfalls erkannt und unterstützt wird. Wir stellen Ihnen die hier stark verkürzt beschriebene Verzeichnisstruktur in Form eines Template-Projektes (siehe Kapitel 5) zur Verfügung!

Quellverzeichnisse

Zentral in einem Softwareprojekt sind natürlich die Quellen (z.B. java-Dateien, aber auch Ressourcen und Property-Dateien). Diese sind separiert nach «produktiven» (main-) Artefakten und «test»-Artefakten:

- **`/src/main/java`** – Produktiver Quellcode
- **`/src/main/resources`** – Ressourcen für produktiven Code.
- **`/src/test/java`** – Testcode; meist auf JUnit basierende Unit- oder Integrationstests.
- **`/src/test/resources`** – Ressourcen für Testfälle.

Das sind somit die vier wichtigsten Verzeichnisse mit denen wir arbeiten. Eine Dokumentation welche weiteren Quell-Verzeichnisse möglich/sinnvoll sind finden Sie hier:

https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

`target`-Verzeichnis

Das Gegenstück zu den Quellen stellen alle Artefakte dar, welche basierend auf den Quellen erzeugt (gebaut) werden können. Zum Beispiel sind das class-Dateien, welche durch die Kompilierung von Java-Dateien entstehen. Oder JAR-Dateien welche durch die Zusammenfassung von class-Dateien erstellt werden. Alle diese Artefakte werden im Verzeichnis `target` (oder einem Unterverzeichnis) abgelegt. Dieses Verzeichnis kann somit jederzeit gelöscht werden, da alle Inhalte vom Buildprozess erstellt werden.

Weitere Verzeichnisse

- **`/src/site`** – In diesem Verzeichnis werden die Quellen (in verschiedenen Formaten) abgelegt, welche von Apache Maven zur Erstellung einer Projekt-Website (**`mvn site`**) herangezogen werden. Mehr Informationen dazu finden Sie bei Apache Maven.
- **`/config`** – Dieses Verzeichnis enthält einige zusätzliche Konfigurationsdateien und entspricht nicht dem Maven-Standard. Es wird aber vom unserem Template-Projekt benötigt.

7. Testing: Verwendung von JUnit

Wenn wir seriöse Softwareentwicklung betreiben wollen, dann ist das Testen von Software untrennbar mit der Entwicklung verknüpft. Um die Effizienz zu steigern, nutzen wir zahlreiche Hilfswerkzeuge, die das Testen (präziser: die Testausführung und Validation) weitgehend automatisieren.

Voraussetzungen

- Ein auf dem HSLU-Template basierendes Projekt.
- In der Entwicklungsumgebung geöffnet / importiert.
- Optional: Apache Maven installiert und in Shell und IDE lauffähig.

Ziele

- Ausführen von auf JUnit basierenden (Unit-)Tests.

Grundlagen

Die Grundlage für die Ausführung von auf JUnit basierenden Tests ist verständlicherweise die Existenz von entsprechenden Testfällen. Das HSLU-Template enthält u.a. eine Beispielklasse **`Point`**, für welche auch eine entsprechende Unit-Testklasse **`PointTest`** vorliegt.

Im Unterricht lernen Sie, wie man solche Testklassen entwickelt. Moderne IDE's stellen Ihnen dazu zahlreiche Vereinfachungen und Hilfsmittel zur Verfügung. Im Folgenden wird nur beschrieben, welche Möglichkeiten Sie haben diese Testfälle auszuführen.

Ausführen aller (Unit-)Testfälle eines Projektes

NetBeans: Sie können über das Hauptmenu (**`Run → Test Project`**) oder das Kontextmenu des Projektes (Test) alle vorhandenen Testfälle ausführen. Das Resultat wird ihnen im «**`Output`**» bzw. unter «**`Test Results`**» angezeigt.

Shell: Verwenden Sie das Kommando **`mvn test`** (schnell, Build failed wenn ein Test misslingt) oder aber generieren Sie mit **`mvn site`** die Website zum Projekt. Letzteres dauert zwar wesentlich länger, dafür finden Sie aber in den **`Project Reports`** der erstellten Website auch eine Menge detaillierter Informationen zu den Ergebnissen.

Ausführen einzelner Testfälle

Das machen Sie bevorzugt in der IDE für die gezielte Fehlersuche.

NetBeans: Sie rufen im Kontextmenu einer Klasse (Testkandidat, z.B. **`Point`**) oder direkt auf der Testklasse (z.B. **`PointTest`**) den Befehl **`Test File`** auf. Auch hier werden die Resultate unter **`Test Results`** angezeigt. Beachten Sie, dass Sie dort auch die Möglichkeit haben die Resultate zu filtern und die Testausführung einfach zu wiederholen.

Zusätzliche Libraries

Das Projekt ist bereits so vorkonfiguriert, dass Ihnen auch die folgenden Zusatzlibraries für (JUnit-)Tests zur Verfügung stehen (für Fortgeschrittene, werden teilweise später noch eingeführt):

- AssertJ - https://joel-costigliola.github.io/assertj/
- EqualsVerifier - https://jqno.nl/equalsverifier/
- Mutability Detector - https://mutabilitydetector.github.io/MutabilityDetector/
- JUnit Pioneer - https://junit-pioneer.org/

Teil 2 – Erweiterte Java Projekte

Erweiterte Funktionen

Module AD, SWDE, VSK und SWDA

8. Softwareaktualisierungen: Java, Entwicklungsumgebung etc.

Spätestens in einem neuen Semester ist es an der Zeit, die Entwicklungsumgebung und alle Werkzeuge wieder auf den neusten Stand zu bringen.

Den grössten Stellenwert hat Java: Oracle veröffentlicht in regelmässigen Abständen Aktualisierungen. Damit werden primär sicherheitsrelevante Fehler korrigiert. Für den Studienalltag sind diese Updates zwar meistens nicht von essenzieller Bedeutung, im Sinne eines professionellen Umganges empfehlen wir Ihnen aber trotzdem die Aktualisierungen jeweils vorzunehmen.

Hinweis: Alle Angaben beziehen sich vorwiegend auf Windows-Systeme.

Voraussetzungen

- Sie haben die Installationen wie in den Kapiteln von Teil 1 beschrieben vorgenommen. Andernfalls adaptieren Sie sinngemäss.

Ziele

- Aktualisierte Versionen des JDK (inklusive der JRE) lauffähig auf Ihrem System installiert.
- Alle wichtigen Enwicklungswerkzeuge sind wieder aktuell.

Java Development Kit und Runtime Environment

Besorgen Sie Sich das aktuelle Java Installationspaket, welches Ihnen auf SWITCHdrive http://bit.ly/2OH3Uhh zur Verfügung gestellt wird. Wir empfehlen eine komplette Deinstallation der alten Java-Version und danach eine Neuinstallation, wie folgend beschrieben!

Deinstallieren Sie zuerst die alte Java-Version (über das Betriebssystem). Kontrollieren Sie danach die Verzeichnisse:

- JDK: **`C:\jdk-17.0.4`** (Beispiel) enthält ggf. noch das manuell kopierte **`/docs`**-Verzeichnis ➔ dieses löschen Sie somit auch **manuell**.
- JRE: **`C:\Programm Files\Java`** – sollte **keine** Unterverzeichnisse (für die eben deinstallierte Version) mehr enthalten.

Nun installieren Sie wie in Kapitel 2 beschrieben Java neu. Die Environment-Variablen **`JAVA_HOME`** und **`Path`** müssen Sie entsprechend an die neue Version anpassen. Bei Updates wird meistens auch die HTML-Dokumentation erneuert. Für einen schnellen, netzunabhängigen Zugriff sollten Sie diese im **`/docs`**-Verzeichnis ebenfalls wieder entpacken (steht ebenfalls auf SWITCHdrive http://bit.ly/2OH3Uhh als ZIP-Datei zur Verfügung!).

Testen Sie danach den Erfolg des Updates durch den Aufruf von **`java -version`**.

Wichtig: Auch in der IDE müssen Sie den Pfad auf das JDK nun ggf. anpassen!

Hinweis: Grundsätzlich können verschiedene Java-(Haupt-)Versionen gleichzeitig installiert sein (z.B. Java 11 und Java 17). Beachten Sie aber, dass die Sie die jeweils als «Default» aktive Version über die Environment-Variablen **`Path`** und **`JAVA_HOME`** festlegen!

Entwicklungsumgebung

Eine grosse Mehrheit der populären Entwicklungsumgebungen verfügt über eigenständige und automatische Update-Mechanismen, so dass die enthaltenen Plugins jeweils auf dem aktuellen Stand sind.

Bei einem Major- oder Minor-Update empfielt sich in vielen Fällen jedoch eine neue (und anfangs parallele) Installation: So haben Sie eine bessere Kontrolle über die Veränderungen, und können neu beurteilen welche zusätzlichen Plugins Sie weiterhin noch verwenden wollen oder müssen.

Leider ist der Zeitaufwand für eine kontrolliere Migration einer liebevoll gepflegten und eingerichteten Entwicklungsumgebung nicht zu unterschätzen! Achten Sie auch darauf, dass bei grösseren Versionswechseln manchmal die Konfigurationen automatisch migriert werden, und es danach ggf. keinen Weg mehr zurück gibt!

Hinweis zu Eclipse JDT: Bei dieser IDE können Sie den Aufwand insofern etwas reduzieren, als dass Sie einen Eclipse-Account einrichten können, über welchen sich ihre Plugins zentral verwalten lassen. So können Sie auch Major-Wechsel mit verhältnismässig geringem Aufwand vollziehen.

Build: Apache Maven

Bei diesen Werkzeugen macht es am meisten Sinn die jeweiligen (Major- und Minor-) Versionen konsequent parallel zu installieren und über die Environment- und IDE-Einstellungen die konkret zu verwendende Version auszuwählen (einmal mehr sind Unix-Benutzer mit der Möglichkeit Links zu erstellen klar im Vorteil ☺).

So können Sie in einem professionellen Umfeld auch ältere Projekte ohne Migrationsaufwand mit der jeweils passenden Version des Buildwerkzeuges weiterhin bauen. Gleichzeitig können Sie jederzeit tatsächlich nicht mehr gebrauchte Versionen schnell und einfach entfernen.

Hinweis: Beachten Sie bitte, dass das lokale Repository im Verzeichnis `.m2` (in Ihrem `HOME`-Verzeichnis) bestehen bleiben kann bzw. soll, insbesondere die darin enthaltene `settings.xml`! Ab und zu kann es aber sinnvoll sein das darin enthaltene Subverzeichnis `repository` zu löschen, um alte Dependencies los zu werden und dadurch viel Platz zu sparen. Die benötigten Dependencies werden danach automatisch wieder vom Repository heruntergeladen.

Maven Template

Mit grosser Wahrscheinlichkeit werden Sie mit jedem Semester auch eine neue Version des von uns zur Verfügung gestellten Maven-Templates bekommen. In der Regel können/sollen Sie das jeweils aktuellste Template aber nur für **neue Projekte** verwenden.

Die Migration von bestehenden, alten Projekten hingegen drängt sich nicht auf und ist höchstens in Ausnahmefällen notwendig. Der einfachste Weg besteht darin, auch für die Migration ein neues Projekt zu erstellen, und dann alle Quellen gezielt zu übernehmen.

9. SCM: Versionskontrolle mit Git

Mit einem Werkzeug zur Versionskontrolle können Sie laufende Veränderungen (Revisionen) schrittweise verfolgen und jederzeit nachvollziehen und auch wieder Rückgängig machen. In Entwicklungsteams sind SCM-Werkzeuge ausserdem eine Grundlage für den gegenseigen Autausch und (ggf. konkurrenzierende) gemeinsame Bearbeitung von Quellcode.

Ziele

- Installation und Konfiguration von Git für die Kommandozeile
- Installation eines grafischen Git-Clients.
- Erstellen und Benutzen eines lokalen Repositories.

Grundlagen

Aufgrund der enormen Popularität von Git, finden sich im Web unzählige Tutorials und Anleitungen zum Thema. Einen guten Einstiegspunkt, inklusive Installation, Konfiguration und Anwendung, finden Sie unter https://githowto.com/

Vorgehen

Optionale Beschreibung eines Vorgehens / Anleitung

a.) Folgen Sie den Links und Anleitungen unter https://githowto.com/ um die notwendige Software zu installieren. Ziel ist es, dass Sie in einer Shell (Eingabeaufforderung) den Befehl

```
git
```

aufrufen können.

b.) Konfigurieren Sie Git mit Ihren Kontaktdaten. Diese Angaben sind später in den Logs von Git-Repositories (History) sichtbar, und dienen zur Identifizierung. Bitte verwenden Sie Ihren Klarnamen und Ihre gültige Email-Adresse, verzichten Sie also auf Fantasie- oder Spassnamen!

```
git config --global user.name "Vorname Nachname "
git config --global user.email "vorname.nachname@hslu.ch"
```

Hinweis: Diese Informationen werden in Ihrem **$HOME**-Verzeichnis in der Datei **.gitconfig** abgelegt und können dort auch jederzeit korrigiert und angepasst werden.

c.) Damit haben Sie bereits alles, um lokal (also ohne zentralen Server) ein Repository zu erstellen und einen Verzeichnisbaum (typisch ein Projekt) zu versionieren. Eine entsprechende Kurzanleitung finden Sie unter: https://githowto.com/create_a_project

d.) Der Einsatz von Git in der Kommandozeile ist zwar sehr effizient, aber nicht immer der einfachste Weg. Seien Sie sich aber bewusst, dass die folgenden Befehle direkt in der Shell und in einem Verzeichnis das bereits ein lokales Repository (**.git**-Verzeichnis) enthält sehr nützlich sind. Probieren Sie sie einfach mal aus!

```
git status
git log
```

e.) Installieren Sie sich einen grafischen Client. Die Auswahl ist mittlerweile sehr gross und es gibt viele Varianten. Wir empfehlen Ihnen einen Client, der in Java implementiert und für alle Plattformen verfügbar ist: **SmartGit** von http://www.syntevo.com/smartgit/.

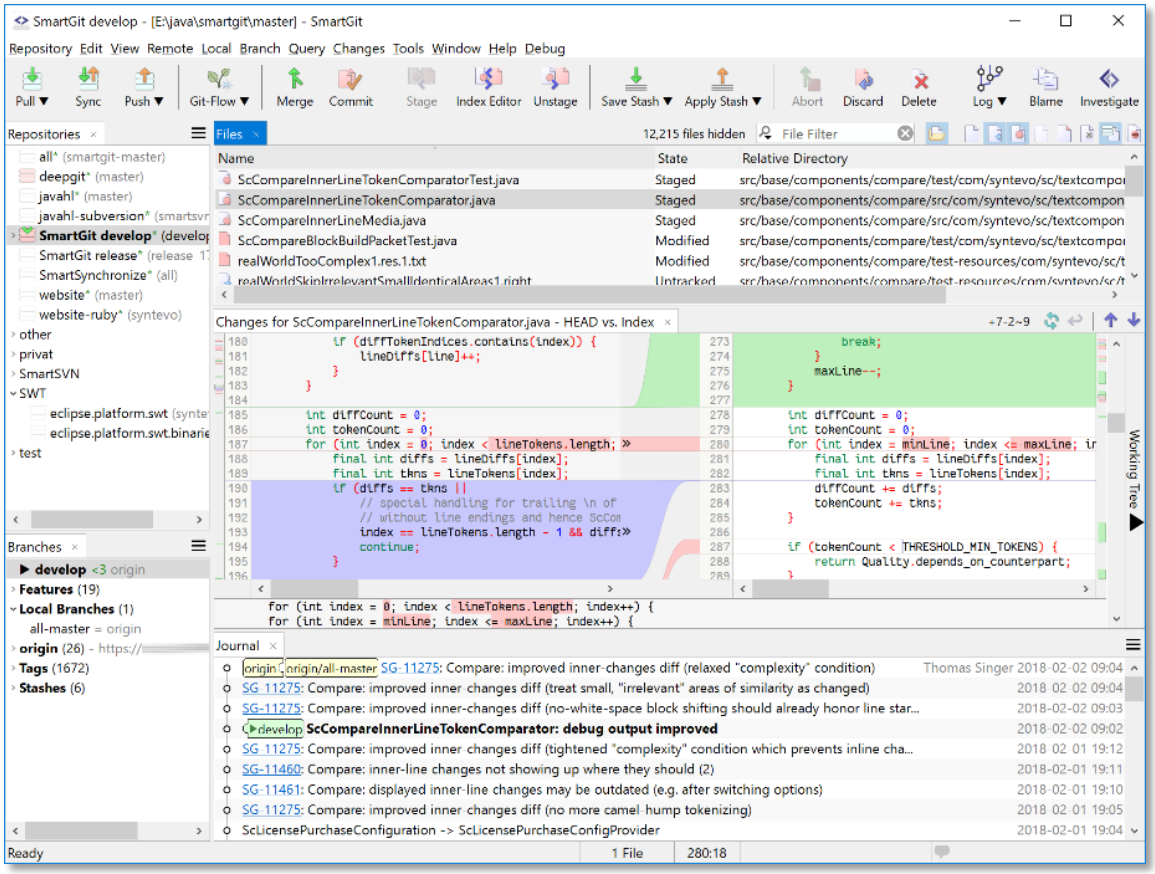

Abbildung 1: SmartGit GUI Client

Er ist für nicht-kommerzielle Nutzung frei verwendbar und relativ leicht bedienbar.

Wichtig: Nicht alle grafischen Git-Clients lesen die Git-Konfigurationen aus **`$HOME/.gitconfig`** ein, sondern verwalten die Identität teilweise selber. Geben Sie auch hier Ihren Klarnamen und Ihre korrekte Email-Adresse an!

e.) Wenn Sie SmartGit (oder einen anderen Client) installiert haben, öffnen Sie damit das mit b) erstellte Projekt/Repository und schauen Sie sich die erstellten Versionen an!

f.) Erstellen Sie auf https://gitlab.enterpriselab.ch/, das vom EnterpriseLab zur Verfügung gestellt wird, unter Ihrem Account ein neues Projekt. Sie erhalten damit direkt eine Anleitung, wie Sie ein bereits vorhandenes, lokales Repository anbinden können. Probieren Sie es aus und nutzen Sie das `push`-Kommando, um Ihr Repository auf den Server hochzuladen.

g.) Nun können Sie mit dem **`clone`**-Befehl das Projekt auf einen beliebigen anderen Rechner «clonen», um es auch dort zu bearbeiten. Lernen Sie Ihren ausgewählten Git-Client kennen und schätzen!