# Praktikum Tooling - Gradle & Classrooms

## Einleitung

In diesem Praktikum machen Sie sich mit dem Build-Tool Gradle vertraut, das zum Erstellen und Verwalten von Java-Projekten verwendet wird. Zudem üben Sie das Bearbeiten und Abgeben von PROG2-Praktikumsaufgaben.

In diesem Praktikum werden Sie:

- Sich in Ihrem GitHub-Classroom registrieren. (Wenn Sie diese Anleitung lesen, ist das bereits erfolgt.)
- Die notwendigen Entwicklerwerkzeuge installieren
- Das Praktikums-Repository aus GitHub-Classrooms auf Ihren Rechner clonen.
- Ein einfaches Java-Projekt mit Gradle erstellen und bearbeiten.
- Das Resultat in GitHub-Classroom abgeben.
- Das Projekt in Ihre bevorzugte IDE integrieren.

### Voraussetzungen

- Ein Benutzerkonto auf GitHub.com

### Tooling

- Installiertes JDK (Version 17)
- Installierter Git-Client
- Ein einfacher Texteditor und Ihre bevorzugte Java IDE
- Kommandozeile / Terminalemulation (java und git muss auf der Kommandozeile funktionieren)

Für Windows empfehlen wir die Git-Bash oder die Bash aus dem Windows Subsystem für Linux (WSL) zu verwenden.
Für Linux & macOS können Sie das mitgelieferte Terminal und Shell (bash, zsh, ...) verwenden.

### Struktur

Ein Praktikum kann verschiedene Arten von Aufgaben enthalten, die wie folgt gekennzeichnet sind:

**[TU] – Theoretische Übung**

Dient der Repetition bzw. Vertiefung des Stoffes aus der Vorlesung und als Vorbereitung für die nachfolgenden Übungen.

**[PU] – Praktische Übung**

Übungsaufgaben zur praktischen Vertiefung von Teilaspekten des behandelten Themas.

**[PA] – Pflichtaufgabe**

Übergreifende Aufgabe zum Abschluss. Das Lösen dieser Aufgaben ist Pflicht. Sie muss bis zum definierten Zeitpunkt abgegeben werden, wird bewertet und ist Teil der Vornote.

### Zeit und Bewertung

Für dieses Praktikum steht rund eine Woche, in den Praktikumslektionen und im Selbststudium, zur Verfügung.
Je nach Kenntnisstand und Erfahrungsstufe benötigen Sie mehr oder weniger Zeit.

Dieses Übungspraktikum wird nicht bewertet.

# 1. Einfaches Projekt mit Gradle erstellen

## 1.1. Gradle auf Ihrem Rechner installieren [PU]

Damit sie Ihre Projekte vollumfänglich mit Gradle verwalten können, muss dies auf Ihrem Rechner installiert sein.

Wenn Sie nur konfigurierte Projekte verwenden, reicht auch der Gradle-Wrapper. Dazu später mehr.

a. Gradle benötigt eine aktuelle Java Installation.
Stellen Sie sicher, dass sie eine aktuelle Version installiert haben [Version 17].

Verifizieren Sie dies in Ihrem Terminal:

```
java --version
```

Ansonsten installieren Sie eine aktuelle Java-JDK-Distribution.

Wir empfehlen Ihnen für die Installation der Entwicklungswerkzeuge ein Package-Manager zu verwenden:

- Chocolatey - Package Manager for Windows
- Homebrew - Package manager for macOS (and Linux)
- Package manager Ihrer Linux-Distribution (APT, YUM/DNF, Pacman, Zypper, Portage, ...)
- SDKMAN - Software Developer Kit Manager (macOS, Linux) ist spezifisch für das Management von Java-spezifischen Werkzeugen gedacht und ermöglicht mehr Kontrolle als die generischen Package-Manager. Neben praktisch allen verfügbaren Java-Distributionen können auch die meisten weiteren Java resp. JVM-basierten Werkzeuge und Frameworks installiert werden (Gradle, Maven, Groovy, Kotlin, Scala, Spring Boot, ...). SDKMAN ermöglicht mehrere Java-Distributionen und Versionen zu installieren und schnell zwischen ihnen zu wechseln.

b. Folgen Sie der offiziellen Anleitung für die Installation von Gradle. Achten Sie darauf, dass Sie eine aktuelle Gradle-Version installieren. Für die Übungen in PROG2 sollten wir die Gradle Version 8.0+ verwenden.
Danach verifizieren Sie im Terminal, dass diese korrekt funktioniert hat:

```
gradle --version
gradle help ①
```

① Beim ersten Ausführen eines gradle Tasks wird der Gradle-Daemon gestartet, um nachfolgende Operationen zu beschleunigen.

## 1.2. Gradle-Projekt in Git-Repository anlegen [PU]

In dieser Übung clonen Sie das Praktikumsrepository und erstellen die Projektstruktur für Gradle.

a. Als Erstes stellen Sie sicher, dass sie Git installiert haben.

Verifizieren Sie dies in Ihrem Terminal:

```
git --version
```

Ansonsten installieren Sie einen aktuellen Git-Client. Siehe dazu Installing Git, bzw. verwenden Sie Ihren Package-Manager.

b. Clonen Sie das Praktikums-Repository in ein PROG2 Arbeitsverzeichnis auf Ihrem Rechner.

Wir empfehlen Ihnen dies auf der Kommandozeile zu tun:

1. Öffnen Sie ein Terminal und navigieren Sie mit dem `cd` Kommando zum gewählten PROG2 Arbeitsverzeichnis.

Statt den Pfad im Terminal von Hand zu tippen, können Sie das Verzeichnis aus dem File-Explorer (Windows) bzw. Finder (macOS) auf das Terminal-Fenster ziehen (drag-n-drop), womit der entsprechende Pfad an der aktuellen Cursor-Position eingefügt wird.

Alternativ kann man auch im Windows File-Explorer ins Verzeichnis Navigieren und im Kontext-Menu (Rechte Maustaste) "Open git bash" auswählen bzw. im macOS-Finder im Kontext-Menu (Dienste > Neues Terminal beim Ordner) auswählen.

2. Kopieren Sie die HTTPS-URL des Praktikumsrepository (bzw. die SSH-Adresse, sofern sie SSH mit GitHub.com konfiguriert haben) vom GitHub-Server und clonen Sie das Repository auf ihren Rechner. In der Kommandozeile geben Sie dazu den folgenden Befehl ein (Verwenden Sie Ihre Repository-URL):

```
git clone <Repository-URL> Lab00-Tooling ①
```

① Mit dem optionalen Parameter `Lab00-Tooling` wird der lokale Verzeichnisname spezifiziert, in welches das Repository gecloned wird. Ansonsten wird der von GitHub-Classroom generierte Name (inklusive Klasse und Benutzername) verwendet.
Wir werden in den nachfolgenden Schritten diesen einheitlichen Namen für das Projektverzeichnis verwenden.

c. Wechseln sie ins Praktikumsverzeichnis (`cd Lab00-Tooling`). Im Moment existiert in diesem Verzeichnis nur diese Anleitung und ein paar Konfigurationsdateien.

d. Als Nächstes erstellen Sie ein Java-Projekt mittels gradle. Der Befehl `gradle init` fragt Sie interaktiv nach den erforderlichen Angaben und erstellt die notwendigen Dateien und Konfigurationen.
In unserem Fall werden wir ein Projekt mit folgenden Attributen erstellen:

| | |
|---|---|
| **Project type** | application |
| **Language** | Java |
| **Subprojects** | no |
| **Configuration-DSL** | Groovy |
| **New API and behavior** | no |
| **Test Framework** | Junit Jupiter |
| **Project name** | `Lab00-Tooling` |
| **Source package** | `ch.zhaw.prog2.tooling` |

Alternativ können diese Angaben auch direkt als Optionen dem Kommando mitgegeben werden:

```
gradle init --type java-application \
  --dsl groovy --test-framework junit-jupiter \
  --project-name Lab00-Tooling \
  --package ch.zhaw.prog2.tooling
```

e. Im Projektverzeichnis wurde die notwendige Struktur und Konfiguration für eine Beispiel-App konfiguriert. Die Struktur finden Sie auch in den Einführungsvorlesung erläutert:

```
Lab00-Tooling
├── .editorconfig ⑨
```

```
├── .gitattributes ⑩
├── .gitignore ⑪
├── .gradle ⑧
│   └── ...
├── README.adoc
├── app ②
│   ├── build.gradle ③
│   ├── src ④
│   │   ├── main
│   │   │   ├── java
│   │   │   │   └── ch
│   │   │   │       └── zhaw
│   │   │   │           └── prog2
│   │   │   │               └── tooling
│   │   │   │                   └── App.java
│   │   │   └── resources
│   │   └── test
│   │       ├── java
│   │       │   └── ch
│   │       │       └── zhaw
│   │       │           └── prog2
│   │       │               └── tooling
│   │       │                   └── AppTest.java
│   │       └── resources
│   └── build ⑤
│       ├── classes
│       │   └── java
│       │       └── main
│       │           └── ch
│       │               └── zhaw
│       │                   └── prog2
│       │                       └── tooling
│       │                           └── App.class
│       └── libs
│           └── app.jar
├── gradle
│   └── wrapper ⑥
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew ⑦
├── gradlew.bat ⑦
└── settings.gradle ①
```

① Projektkonfiguration. Sie enthält den Projektnamen und die Liste der Module/Subprojekte. Liegt immer im Root-Ordner des Projektes.

② Unterverzeichnis mit Anwendung/Modul. Der Name muss mit Angabe in Projektkonfiguration übereinstimmen.

③ Konfiguration der Anwendung/Modul (Dependencies, Plugins, …)

④ Quellcode der Anwendung/Modul, getrennt nach Anwendung (`main`), Tests (`test`), jeweils unterteilt in Klassen die übersetzt (`java`) und Ressourcen, die nur kopiert werden (`resources`).

⑤ Im `build`-Verzeichnis liegen die generierten Artefakte, wie z.B. die kompilierten Klassen (`classes`) oder die finalen Anwendungsarchive (`libs`). Da alle Dateien wieder generiert werden können, sollte dieses Verzeichnis nicht ins Git-Repository (→ `.gitignore`)

⑥ Gradle-Wrapper-Archiv und -Konfiguration (siehe Abschnitt Der Gradle-Wrapper).

⑦ Scripts zum Ausführen des Gradle-Wrappers (siehe Abschnitt Der Gradle-Wrapper)

⑧ Gradle-Cash-Verzeichnis (`.gradle`, in welche heruntergeladene Elemente und Statusinformationen abgelegt werden. Nicht unter Versionskontrolle (→ `.gitignore`).

⑨ Die EditorConfig-Konfiguration wurde nicht durch gradle erstellt, sondern im

Praktikumsrepository mitgeliefert. Sie erlaubt die Coding-Styles (Einrückungen, Zeilenende, etc.) IDE unabhängig im Projekt zu speichern. Die IDE berücksichtigen diese Konfiguration (teilweise wird ein Plugin benötigt).

⑩ `.gitattributes` wird erstellt, falls es nicht schon existiert, damit Windows Batch-Dateien (z.B. `gradle.bat`) korrekt im Git-Repository kodiert werden.

⑪ In `.gitignore` müssen die Verzeichnisse für die generierten bzw. Cache-Dateien konfiguriert sein. Falls die Datei nicht existiert, wird sie durch `gradle init` erstellt.

Die im Praktikum enthaltene `.gitignore`-Datei beinhaltet weitere Muster von Dateien, die nicht ins Repository eingecheckt werden sollen, wie zum Beispiel die IDE-Konfigurationen (`.idea`, `.project`, `.vscode`, `bin`, ...), andere temporäre Dateien wie `*.log`, `.DS_Store`, `.Thumbs` und viele mehr.

f. Starten Sie die Tests der Anwendung.

*Linux, macOS, git-bash, WSL:*

```
./gradlew test ①
```

*Windows:*

```
.\gradlew.bat test ①
```

① Es wird empfohlen, den Gradle-Wrapper zum Ausführen von Gradle-Tasks zu verwenden. Mehr dazu im Abschnitt Der Gradle-Wrapper.

Automatisch wird die Anwendung und die Tests kompiliert, sowie die Tests via den konfigurierten Test-Runner ausgeführt.

g. Da es sich um eine Anwendung handelt, kann diese auch mit gradle gestartet werden:

*Linux, macOS, git-bash, WSL:*

```
./gradlew run
```

*Windows:*

```
.\gradlew.bat run
```

h. Ändern Sie die Begrüssung in der App mit einem Texteditor von `"Hello World!"` nach z.B. `"Hello PROG2!"` und starten Sie die Anwendung.
Sie werden feststellen, dass gradle merkt, dass die Quelle geändert wurde (neuer ist als die Klasse) und die notwendigen Klassen automatisch kompiliert werden, bevor sie gestartet wird.

i. Mit `./gradlew tasks` werden die verfügbaren Tasks angezeigt. Zum Beispiel können Sie mittels `./gradlew clean` automatisch das Projekt bereinigen und alle generierten Dateien (`build` -Verzeichnis) löschen. Oder mit `./gradlew javadoc` werden die JavaDoc HTML-Seiten in `app/build/docs/javadoc` generiert.
Mit `./gradlew help --task <Taskname>` werden zusätzliche Informationen (z.B. Optionen) für einen spezifischen Task ausgegeben.

j. Ein weiterer Task der für Java-Projekte interessant ist, ist das Auflisten der Abhängigkeiten (Dependencies), d.h. die benötigten Libraries. Dies wird mit dem Task `dependencies` ermöglicht. Wenn Sie diesen auf dem Root-Projekt aufrufen, wird er Ihnen aber nichts anzeigen:

```
$ ./gradlew dependencies

> Task :dependencies

------------------------------------------------------------
Root project 'Lab00-Tooling'
------------------------------------------------------------
```

```
No configurations
```

Der `dependencies` Task wird nicht automatisch an alle Subprojekte delegiert, wie zum Beispiel bei `build`, `assemble`, `clean`, `javadoc` oder `run`.
Das heißt, Sie müssen den Task für das spezifische Subprojekt/Modul (in unserem Fall `app`) direkt aufrufen:

```
$ ./gradlew :app:dependencies

> Task :app:dependencies

------------------------------------------------------------
Project ':app'
------------------------------------------------------------

annotationProcessor - Annotation processors and their dependencies for source set
'main'.
No dependencies

compileClasspath - Compile classpath for source set 'main'.
\--- com.google.guava:guava:31.1-jre
     +--- com.google.guava:failureaccess:1.0.1
     +--- com.google.guava:listenablefuture:9999.0-empty-to-avoid-conflict-with-guava
     +--- com.google.code.findbugs:jsr305:3.0.2
     +--- org.checkerframework:checker-qual:3.12.0
     +--- com.google.errorprone:error_prone_annotations:2.11.0
     \--- com.google.j2objc:j2objc-annotations:1.3
...
```

Es werden die direkten und transitiven[1] Abhängigkeiten für die verschiedenen Phasen (compile, runtime) und Source Sets (main, test) angezeigt.

Wenn Sie mehrere Subprojekte mit lauffähigen Anwendungen haben (`run` Task vorhanden), so startet run auf der Root-Projektebene je eine Instanz pro solchem Subprojekt. Das ist nicht immer, was gewünscht ist. In diesen Fällen macht es oft Sinn den Task für die einzelnen Teilanwendungen direkt zu starten (z.B. `./gradlew :app:run`)

### Der Gradle-Wrapper

Beim Erstellen des Projekts mit `gradle init` wurde auch der Gradle-Wrapper installiert (`gradle wrapper`). Damit wird Gradle als kompiliertes Archiv `gradle-wrapper.jar` im Projekt abgelegt, welches mit einem Shell-Script `gradlew` für Linux, macOS, git-bash, WSL und einem Batch-Script `gradlew.bat` für die Windows-Kommandozeile (CMD/PowerShell) ausgeführt werden kann. Gradle wird so mit dem Projekt ausgeliefert und muss nicht im System installiert sein.
Mittels der Wrapper-Konfiguration `gradle-wrapper.properties`, wird damit auch spezifiziert, welche Gradle-Version für dieses Projekt verwendet werden soll.

Weitere Informationen finden Sie in der Gradle Wrapper Dokumentation.

## 1.3. Abgeben des Resultats in GitHub Classroom [PU]

Während Sie an den Praktikumsaufgaben arbeiten sollten Sie die Resultate regelmässig ins Repository einchecken. Dies erfolgt mit den bekannten Git-Befehlen.

Wir empfehlen Ihnen Git auf der Kommandozeile zu verwenden, um die Anwendung der Kommandos zu üben.

a. Fügen Sie die neuen Dateien zur Staging-Area hinzu

```
git add --all
```

b. Verifizieren sie, dass keine neuen Dateien versioniert werden, die nicht ins Repository gehören:

```
git status
```

Gegebenenfalls fügen Sie diese zu `.gitignore` hinzu und entfernen diese mittels `git restore --staged <Datei>` aus der Staging-Area.

c. Sobald alles korrekt aussieht, commiten Sie die Änderungen zum Repository:

```
git commit -m "Created initial gradle project"
```

In regelmässigen Abständen sollten Sie Ihre Lösungen zum GitHub-Repository transferieren (Pushen). Zum Beispiel, sobald Sie einen Abschnitt des Praktikums gelöst haben. Auf jeden Fall müssen Sie jedoch Ihre Lösung vor dem Abgabetermin des Praktikums übermitteln.

d. Übermitteln Sie alle neuen Commits zum GitHub-Repository:

```
git push origin
```

Sie werden feststellen, dass nach dem Push automatisch auf GitHub ein Autograding-Prozess gestartet wird. Ob dieser erfolgreich war, wird ihnen angezeigt. Sie können jedoch auch die Details in Ihrem GitHub-Praktikumsrepository unter "Actions" inspizieren und nachsehen, warum ein Test fehlgeschlagen ist.

Für die Autograding-Funktion existieren Nutzungslimiten. Bitte pushen Sie Ihre Lösungen deshalb nicht unnötig oft, sondern nur dann, wenn Sie sinnvolle Resultate haben. Zum Beispiel, wenn Sie eine Pflichtaufgabe gelöst haben und die lokalen Tests nicht mehr Fehlschlagen. Und natürlich den finalen Stand vor dem Abgabetermin.

# 2. Integration in die IDE

## 2.1. Importieren und Bearbeiten des Projekts in Ihrer bevorzugten IDE [PU]

Die verbreiteten Java-IDEs unterstützen Gradle basierte Projekte, erkennen dies und importieren die Projektkonfiguration automatisch bzw. verwenden Gradle im Hintergrund für das Management des Projektes.

a. Öffnen Sie in Ihrer bevorzugten IDE gemäss den untenstehenden Instruktionen und Hinweisen.
b. Inspizieren Sie die Projektstruktur, Konfigurationsdateien, etc.
c. Wo finden Sie die Gradle-Hilfsmittel in ihrer IDE? Wie können Sie einen spezifischen Gradle-Task aus der IDE starten?
d. Führen Sie die Anwendung aus.
e. Führen Sie den Test aus. Wo werden die Resultate angezeigt?

### JetBrains IntelliJ IDEA

Das Gradle-Plugin ist in IntelliJ standardmässig enthalten. Es ist so konfiguriert, dass es Gradle-Projekte im Hintergrund automatisch den Gradle-Wrapper verwenden, falls vorhanden.

1. Öffnen Sie Ihr Projekt mittels *File › Open...* und wählen Ihr Projektverzeichnis aus.

Wenn Sie das IDEA-Kommandozeilen-Werkzeug von IntelliJ installiert haben (JetBrains Toolbox), können Sie das Projekt auch aus dem Terminal mittels `idea <Verzeichnisname>` öffnen.

Es wird automatisch erkannt, dass es sich um ein Gradle-Projekt handelt und die Gradle-

Konfiguration übernommen. In der Projektstruktur (*File › Project Structure…*) kann dies verifiziert werden.

2. Im *Gradle-Tool* Fenster (Knopf am rechten Rand) werden automatisch alle Module und Tasks angezeigt und können von dort ausgeführt werden. Ebenfalls kann hier die Gradle-Konfiguration synchronisiert werden, wenn diese in den Konfigurationsdateien (`settings.gradle`, `build.gradle`, …) angepasst wird. Details finden Sie in der Dokumentation zum Gradle Tool Fenster auf der Jetbrains Webpage

### Eclipse

Für Eclipse wird das Eclipse Buildship Plugin für die Verwendung von gradle benötigt. Je nach Distribution ist dieses Plugin bereits installiert.

1. Öffnen Sie den Marketplace (*Help › Marketplace*) und suchen Sie nach *buildship*
2. Installieren Sie das Plugin und starten Eclipse neu.
3. Öffnen Sie das Projekt mittel *File › Open Projects from File System*
4. Wählen Sie *Show other specialized import wizards*
5. Selektieren Sie *Gradle* / *Existing Gradle Project*

Sie finden die **gradle tasks** in der view *Gradle Tasks*. Diese können Sie im Menu *Window › Show View › Other… › Gradle Tasks* aktivieren.

Weitere Informationen finden Sie in der Dokumentation zum Eclipse Buildship Plugin.

### Visual Studio Code

In Visual Studio Code benötigen Sie das Gradle for Java Plugin, neben Microsofts Extension Pack for Java zur generellen Unterstützung von Java.

1. Suchen Sie im Extension-Marketplace nach den Plugins und installieren sie diese.
2. Öffnen Sie das Projekt mittels *Datei › Ordner öffnen…*

Wenn Sie das VSCode-Kommandozeilen-Werkzeug installiert haben (*Anzeigen › Befehlspalette…*, nach "Install code command in Shell" suchen und ausführen), können Sie das Projekt auch aus dem Terminal mittels `code <Verzeichnisname>` öffnen.

Das Gradle-Projekt wird automatisch erkannt und konfiguriert.

3. Die Gradle Projektübersicht und Tasks finden sie im Gradle-Fenster (linker Rand)

# Abschluss

Damit sind Sie am Ende des Praktikums angelangt. Stellen Sie sicher, dass Ihre Sie Lösungen vor dem Abgabetermin ins GitHub-Classrooms Abgaberepository pushen.

## Hinweis zu den nachfolgenden Praktika

In den kommenden bewerteten Praktika werden wir Ihnen die Gradle-Konfiguration bereits im Repository mitliefern. Sie müssen die Gradle-Projekte nicht mehr erstellen, sondern können diese direkt in der IDE öffnen und bearbeiten.

Hingegen müssen Sie wissen, wie man die Projekte bearbeitet, die Anwendungen / Tasks startet und gegebenenfalls die Konfiguration erweitert oder anpasst.

[1] Rekursive Abhängigkeiten der Libraries