Modul 324 Notes

Table of Contents

1 Modul 324 Notes

2 Was ist DevOps

3 Java Modules

- 3.1 Java Modules
- 3.2 Multi Module

https://bbw-it.github.io/324_main_rupe/03_Drehbuch/Drehbuch_Modul324_HS26_5IA23b_PR/

2 Was ist DevOps

Was gehört alles zu Dev-Ops? - Continous Circle / Lifecyvle - Pipeline - Shared responsibility - Softwareentwicklung - Betrien {zusammen} - Enless loop - IDE's - Automatisierung CI/CD - IaC (Infra) - Vulability Test - Mean Time to Recovery (MTTR)

2.1 Fragen beantworten

1) Wieso macht man DevOps?
DevOps verbindet Entwicklung (Development) und Betrieb (Operations), um Software schneller, zuverlässiger und mit weniger Fehlern bereitzustellen.
Um die Prozesse zwischen Softwareentwicklung & operationalen IT-Teams (Betrieb) zu optimieren die Barrieren zwischen traditionell isolierten Entwicklungs- und operationellen IT-Teams zu beseitigen.

---

2) Was hat das Management davon (was profitieren Sie)?

- Höhere Geschwindigkeit: Neue Funktionen können schneller veröffentlicht werden.
- Hoche Wirtschaftlichkeit: Weniger Fehler und Ausfälle --> geringere Kosten.
- Geringe Risiko: Kleinere Realeases + automatisierte Tests + Starkes Monitoring.
- Erhöht die Wettbewerbsfähigkeit.

---

3) Was haben die Entwickler davon (was profitieren Sie)?

- Identische Umgebungen: macht aus das keine böse Überaschungen erscheinen, geringere Fehlerrisiko. (Weniger "Bei mir funktioniert es nicht" probleme)
- Automatisierte Tests, CI/CD: Schnelleres Feedback.
- Kleinere Releases: Fehler leichter finden und beheben.
- Weniger Angst vor Deployments.

---

4) Welche Vorgaben an das DevOps lassen sich daraus ableiten? - So viel wie möglich automatisieren + Autmatisierte Tests - Kleine, atomare Releases - Umfassendes Monitoring und Logging - schnelles Feedback/ Validierug

---

5) Notiere 5 Anforderungen an DevOps, die sich aus dem obigen ergeben? 1. Vollständige Automatisierung der Realse-Pipeline 2. Automatisierte Tests vor dem Deployment 3. Kleine und häufige Releases 4. Umfassende Monitoring & Logging 5. Schnelle Wiederherstellung bei Fehlern

3 Java Modules

3.1 Java Modules

3.1.1 Arten von Java-Splitting

- **Buildmanager:**
- **Maven:** Zentrale `pom.xml`-Konfiguration für Abhängigkeiten, spezifisch für Java gemacht
- **Gradle:** Unterstützt mehrere Sprachen, aber sehr gut für Java; nutzt Groovy- oder Kotlin-DSL (Domain Specific Language) zur Konfiguration
- **Apache Ant:** Älter als Maven und Gradle, nutzt ebenfalls XML (`build.xml`) zur Konfiguration; wird wegen Einfachheit und Direktheit weiterhin benutzt
- **Java Platform Module System (JPMS/Jigsaw)** *(verstehe ich noch nicht ganz)*: Seit Java 9; erlaubt es, Code auf Sprachebene in Module aufzuteilen. Jedes Modul deklariert in `module-info.java`, welche Pakete es exportiert und welche Module es benötigt – interne Pakete bleiben für andere unsichtbar.

```
module meine.app {
    requires java.sql;          // Ich brauche das SQL-Modul
    exports com.beispiel.api;   // Dieses Paket dürfen andere benutzen
    // Alle anderen Pakete bleiben versteckt, auch wenn Klassen public sind!
}
```

- **Git Submodule:** Wenn ein Projekt aus mehreren Git-Repos besteht, kann man diese als Submodule einbinden; erlaubt die Bearbeitung als einzelnes Projekt in der IDE, obwohl es auf unterschiedlichen Repos gehostet wird
- **Docker und Microservices:** Jede Komponente als eigener Service; dies fördert die Isolation, erleichtert das Deployment und ermöglicht eine effiziente Skalierung

IDEs

IDEs unterstützen die Entwicklung in vielen verschiedenden System sowie autoamtische Task generierenung basierend auf den Config Dateien. Zudem erlauben IDEs heutztzutage auch einfach Code zu spliten.

3.1.2 Sources

- Java-Projekt und Artefakte
- Möglichkeiten, ein Maven-Projekt zu modularisieren

3.2 Multi Module Maven Projekt und Git-Submodul

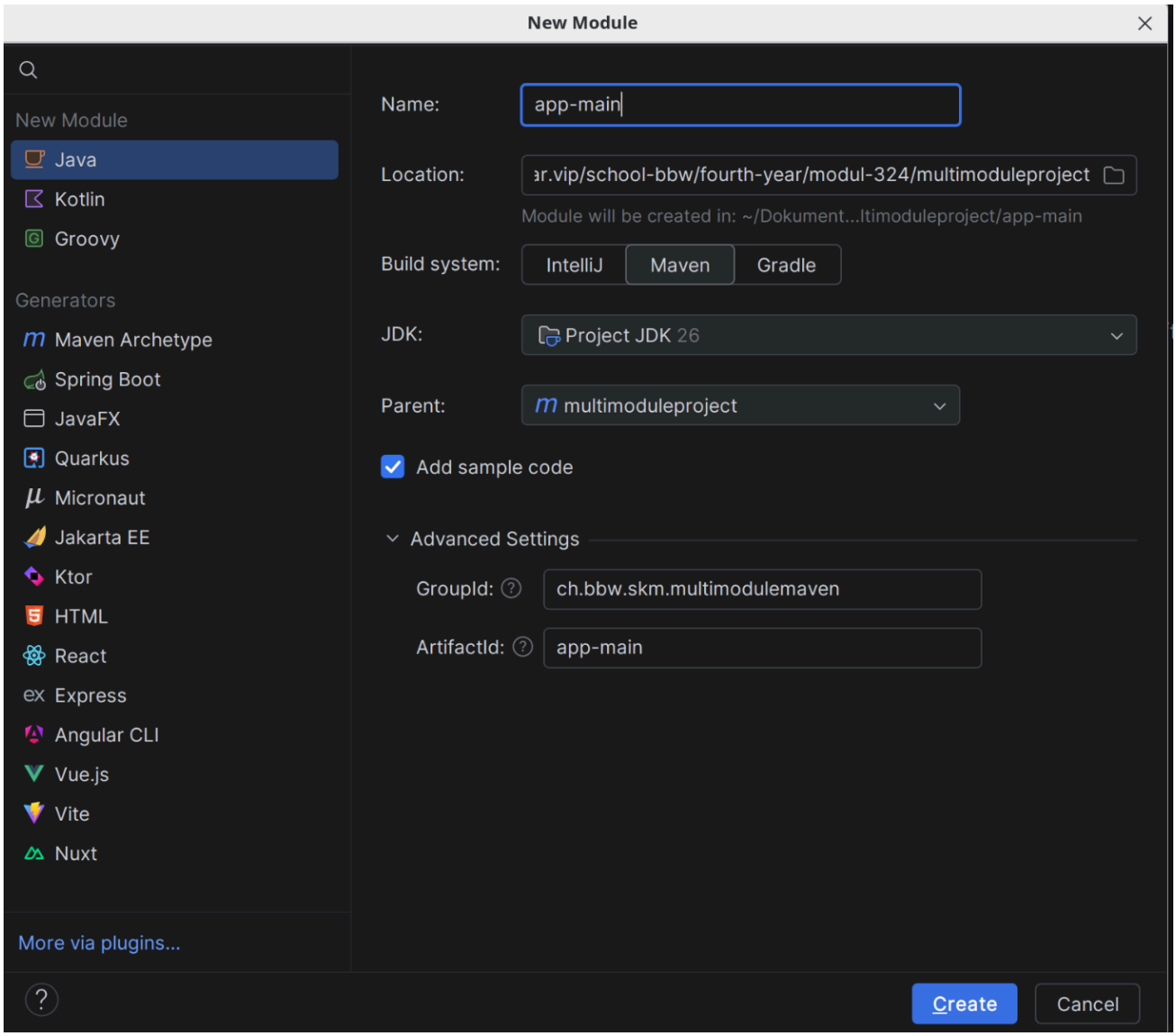
3.2.1 Maven Submodules

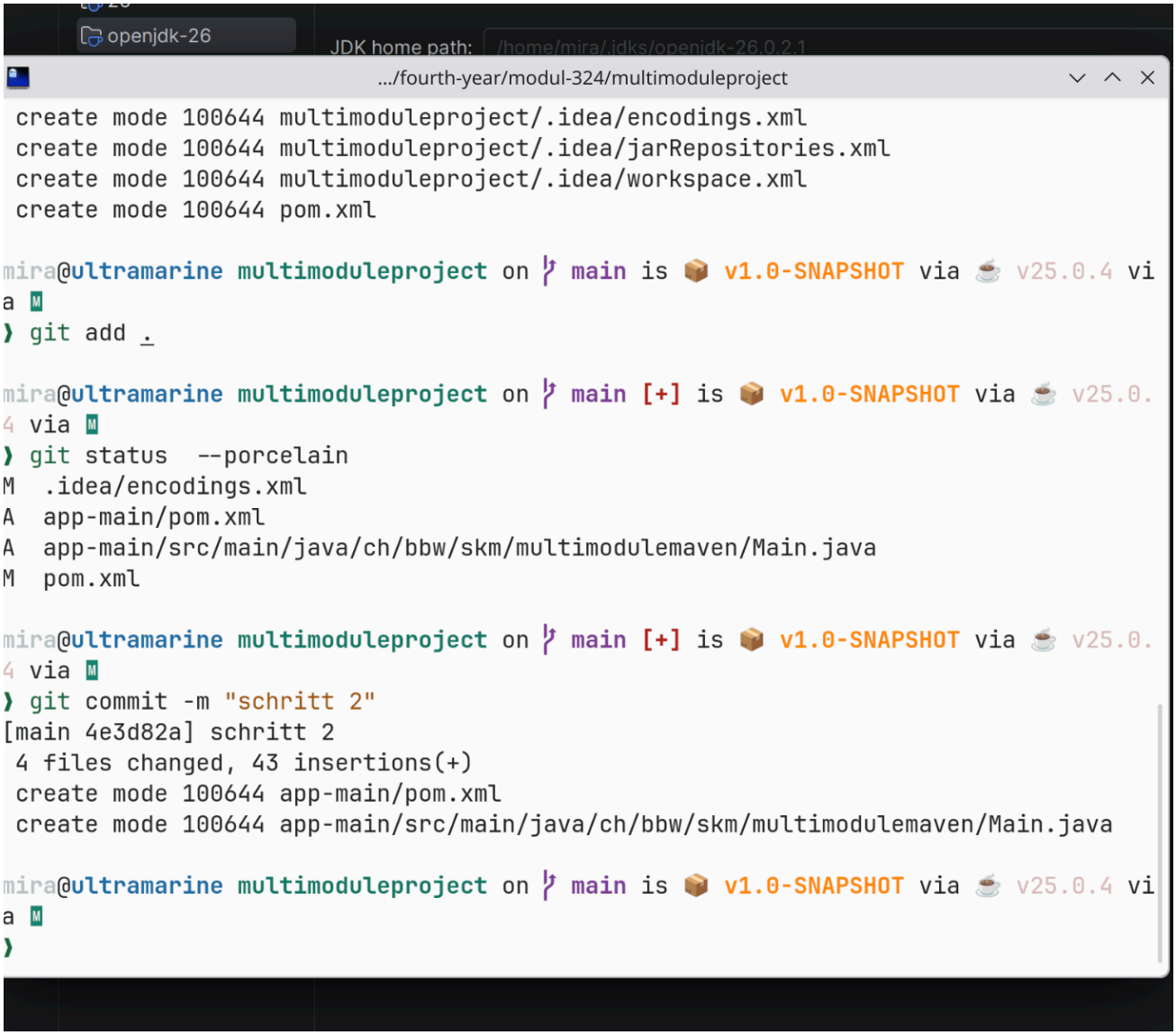
Neues Maven Submodule in Idea erstellen: `-> File -> New -> Module -> Maven`

Schritt 1

```
mira@ultramarine multimoduleproject on  main [+] is 📦 v1.0-SNAPSHOT via ☕ v25.0.
4 via M
❯ git commit -m "init"
[main (Root-Commit) 3b5211e] init
 10 files changed, 214 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/encodings.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/vcs.xml
 create mode 100644 multimoduleproject/.idea/compiler.xml
 create mode 100644 multimoduleproject/.idea/encodings.xml
 create mode 100644 multimoduleproject/.idea/jarRepositories.xml
 create mode 100644 multimoduleproject/.idea/workspace.xml
 create mode 100644 pom.xml
```

Scrhit 2

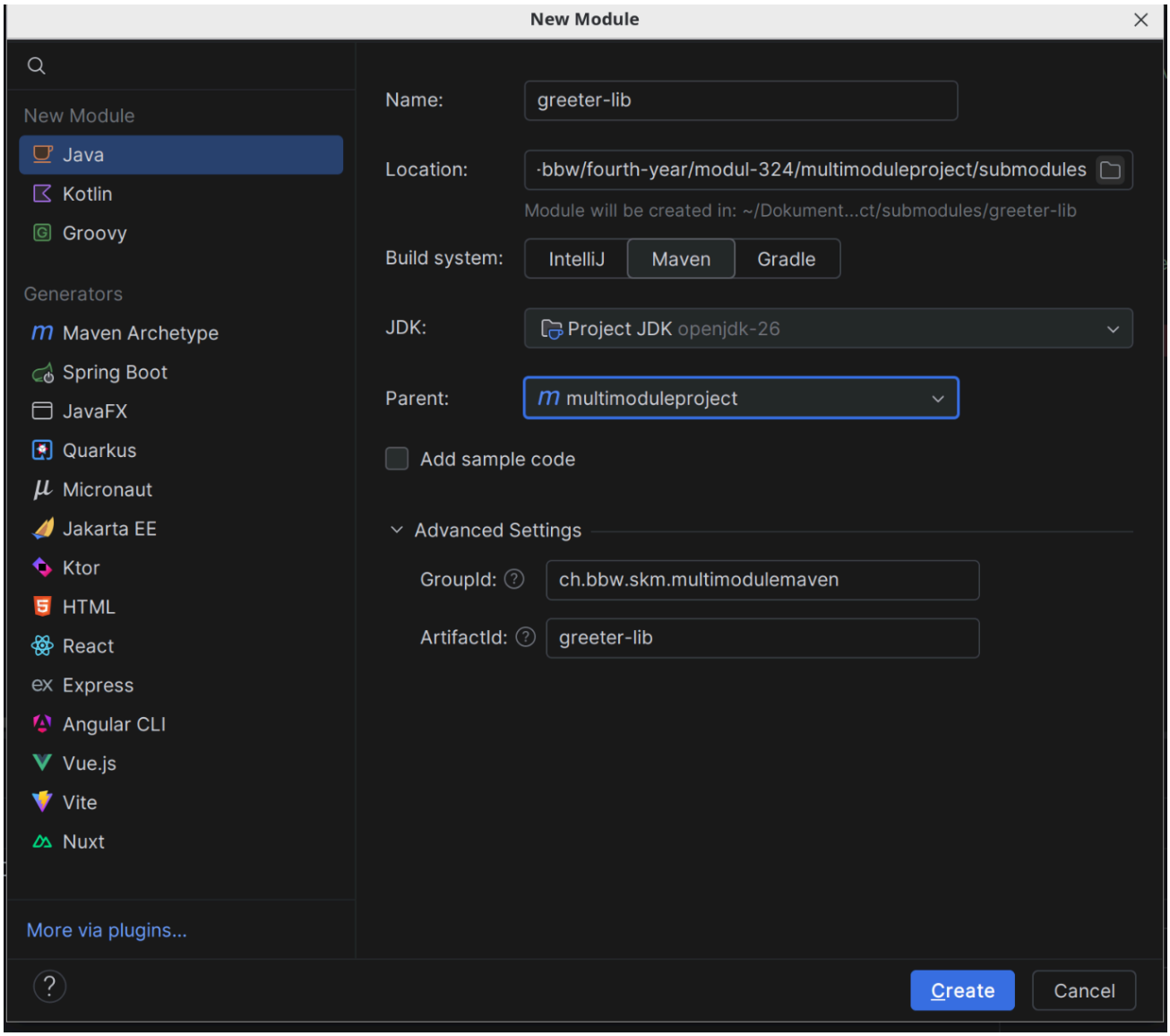

New Module
New Module
Java
Kotlin
Groovy
Generators
Maven Archetype
Spring Boot
JavaFX
Quarkus
Micronaut
Jakarta EE
Ktor
HTML
React
Express
Angular CLI
Vue.js
Vite
Nuxt
More via plugins...
Name:
app-main
Location:
ar.vip/school-bbw/fourth-year/modul-324/multimoduleproject
Module will be created in: ~/Dokument...ltimoduleproject/app-main
Build system:
IntelliJ
Maven
Gradle
JDK:
Project JDK 26
Parent:
multimoduleproject
Add sample code
Advanced Settings
GroupId:
ch.bbw.skm.multimodulemaven
ArtifactId:
app-main
Create
Cancel


openjdk-26
JDK home path:
/home/mira/.jdks/openjdk-26.0.2.1
.../fourth-year/modul-324/multimoduleproject
create mode 100644 multimoduleproject/.idea/encodings.xml
create mode 100644 multimoduleproject/.idea/jarRepositories.xml
create mode 100644 multimoduleproject/.idea/workspace.xml
create mode 100644 pom.xml
mira@ultramarine multimoduleproject on main is v1.0-SNAPSHOT via v25.0.4 via M
git add .
mira@ultramarine multimoduleproject on main [+] is v1.0-SNAPSHOT via v25.0.4 via M
git status --porcelain
M .idea/encodings.xml
A app-main/pom.xml
A app-main/src/main/java/ch/bbw/skm/multimodulemaven/Main.java
M pom.xml
mira@ultramarine multimoduleproject on main [+] is v1.0-SNAPSHOT via v25.0.4 via M
git commit -m "schritt 2"
[main 4e3d82a] schritt 2
4 files changed, 43 insertions(+)
create mode 100644 app-main/pom.xml
create mode 100644 app-main/src/main/java/ch/bbw/skm/multimodulemaven/Main.java
mira@ultramarine multimoduleproject on main is v1.0-SNAPSHOT via v25.0.4 via M

Output geht 👍

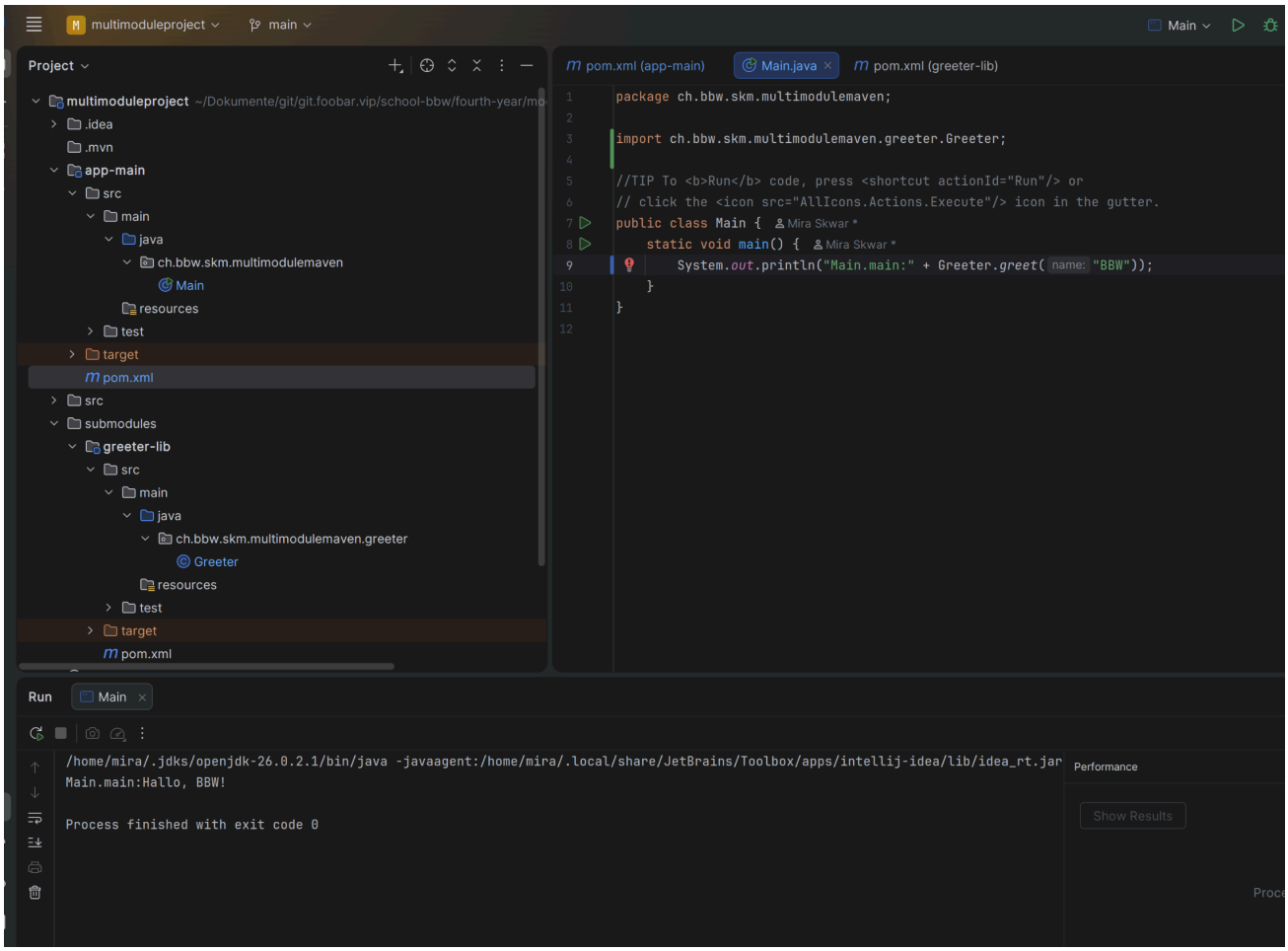

Run
Main
/home/mira/.jdks/openjdk-26.0.2.1/bin/java -javaagent:/home/mira/.local/share/JetBrains/
Hello and welcome!
i = 1
i = 2
i = 3
i = 4
i = 5
Process finished with exit code 0
multimoduleproject > app-main > src > main > java > ch > bbw > skm > multimodulemaven > Main

Schritt 3



> [!WARNING] Default mässig wird eine Java Klasse in Idea ohne package angabe gemacht dadruch hatte Schritt 4 probleme Oben folgendes hinzufügen: `package ch.bbw.skm.multimodulemaven.greeter;`

Schritt 4


multimoduleproject
main
Main
Project
multimoduleproject ~/Dokumente/git/git.foobar.vip/school-bbw/fourth-year/mo
.idea
.mvn
app-main
src
main
java
ch.bbw.skm.multimodulemaven
Main
resources
test
target
pom.xml
src
submodules
greeter-lib
src
main
java
ch.bbw.skm.multimodulemaven.greeter
Greeter
resources
test
target
pom.xml
pom.xml (app-main)
Main.java
pom.xml (greeter-lib)
package ch.bbw.skm.multimodulemaven;
import ch.bbw.skm.multimodulemaven.greeter.Greeter;
//TIP To Run code, press <shortcut actionId="Run"/> or
// click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.
public class Main {
Mira Skwar *
static void main() {
Mira Skwar *
System.out.println("Main.main:" + Greeter.greet(name: "BBW"));
}
}
Run
Main
/home/mira/.jdks/openjdk-26.0.2.1/bin/java -javaagent:/home/mira/.local/share/JetBrains/Toolbox/apps/intellij-idea/lib/idea_rt.jar
Main.main:Hallo, BBW!
Process finished with exit code 0
Performance
Show Results

Schritt 5

pom.xml (app_main) | Main.java | pom.xml (greeter-lib) | Greeter.java

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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> Compatible with Maven 3
    <parent>
        <groupId>ch.bbw.skm.multimodulemaven</groupId>
        <artifactId>multimoduleproject</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <artifactId>app_main</artifactId>
    <packaging>jar</packaging>
    <name>app_main</name>

    <dependencies>
        <dependency>
            <groupId>ch.bbw.skm.multimodulemaven</groupId>
            <artifactId>greeter-lib</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

</project>
```

pom.xml (app_main) | Main.java | pom.xml (greeter-lib) | Greeter.java

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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> Compatible with Maven 3
    <parent>
        <groupId>ch.bbw.skm.multimodulemaven</groupId>
        <artifactId>multimoduleproject</artifactId>
        <version>1.0-SNAPSHOT</version>
        <relativePath>../../pom.xml</relativePath>
    </parent>

    <artifactId>greeter-lib</artifactId>
    <packaging>jar</packaging>
    <name>greeter-lib</name>

</project>
```

Schritt 6

```
.../fourth-year/modul-324/multimoc
Verzeichnis 'submodules/greeter-lib/src/main' wurde entfernt
Verzeichnis 'submodules/greeter-lib/src/test/java' wurde entfernt
Verzeichnis 'submodules/greeter-lib/src/test' wurde entfernt
Verzeichnis 'submodules/greeter-lib/src' wurde entfernt
'submodules/greeter-lib/target/classes/ch/bbw/skm/multimodulemaven/greeter/Greeter.class' wurde e
Verzeichnis 'submodules/greeter-lib/target/classes/ch/bbw/skm/multimodulemaven/greeter' wurde ent
Verzeichnis 'submodules/greeter-lib/target/classes/ch/bbw/skm/multimodulemaven' wurde entfernt
Verzeichnis 'submodules/greeter-lib/target/classes/ch/bbw/skm' wurde entfernt
Verzeichnis 'submodules/greeter-lib/target/classes/ch/bbw' wurde entfernt
Verzeichnis 'submodules/greeter-lib/target/classes/ch' wurde entfernt
Verzeichnis 'submodules/greeter-lib/target/classes' wurde entfernt
Verzeichnis 'submodules/greeter-lib/target/generated-sources/annotations' wurde entfernt
Verzeichnis 'submodules/greeter-lib/target/generated-sources' wurde entfernt
Verzeichnis 'submodules/greeter-lib/target' wurde entfernt
Verzeichnis 'submodules/greeter-lib' wurde entfernt
Verzeichnis 'submodules/' wurde entfernt

mira@ultramarine multimoduleproject on  main [X] is 📦 v1.0-SNAPSHOT via ☕ v25.0.4 via M
❯ ll
insgesamt 4
drwxr-xr-x. 1 mira mira  32 28. Aug 14:51 app-main
-rw-r--r--. 1 mira mira 817 28. Aug 14:40 pom.xml
drwxr-xr-x. 1 mira mira  16 28. Aug 14:28 src

mira@ultramarine multimoduleproject on  main [X] is 📦 v1.0-SNAPSHOT via ☕ v25.0.4 via M
❯ rm -rfv app-main/target

mira@ultramarine multimoduleproject on  main [X] is 📦 v1.0-SNAPSHOT via ☕ v25.0.4 via M
❯ git restore submodules/

mira@ultramarine multimoduleproject on  main [X] is 📦 v1.0-SNAPSHOT via ☕ v25.0.4 via M
❯ rm -rfv app-main/target src
Verzeichnis 'app-main/target/generated-sources/annotations' wurde entfernt
Verzeichnis 'app-main/target/generated-sources' wurde entfernt
'app-main/target/classes/ch/bbw/skm/multimodulemaven/Main.class' wurde entfernt
Verzeichnis 'app-main/target/classes/ch/bbw/skm/multimodulemaven' wurde entfernt
Verzeichnis 'app-main/target/classes/ch/bbw/skm' wurde entfernt
Verzeichnis 'app-main/target/classes/ch/bbw' wurde entfernt
Verzeichnis 'app-main/target/classes/ch' wurde entfernt
Verzeichnis 'app-main/target/classes' wurde entfernt
Verzeichnis 'app-main/target' wurde entfernt
Verzeichnis 'src/main/java' wurde entfernt
Verzeichnis 'src/main/resources' wurde entfernt
Verzeichnis 'src/main' wurde entfernt
Verzeichnis 'src/test/java' wurde entfernt
Verzeichnis 'src/test' wurde entfernt
Verzeichnis 'src' wurde entfernt
```

> [!NOTE] Ausversehen `submodules` dir gelöscht zumglück in git gehabt.

Fragen zum Verständnis?

- Was ist ein Multi-Modul-Maven-Projekt? Es ist ein Projekt wo aus mehreren Maven Subprojekten besteht.
- Was ist die Rolle des Parent Projektes? Es Bundeled und Unified alle Subpackages
- Was ist die Rolle der Module? Es entkapselt seinen relavanten Code in sich.
- Wie werden die Module in der pom.xml des Parent Projektes definiert? Gar nicht

- Wie werden die gemeinsamen Einstellungen im Parent Projekt definiert? Unter dem `properties` Key im `pom.xml`
- Wie wird das Parent Projekt in den Modulen referenziert?

```
    <parent>
        <groupId>ch.bbw.skm.multimodulemaven</groupId>
        <artifactId>multimoduleproject</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
```

- Wie wird die Abhängigkeit von einem Modul zu einem anderen Modul definiert? Mit einem `dependency` entry

```
    <dependencies>
        <dependency>
            <groupId>ch.bbw.skm.multimodulemaven</groupId>
            <artifactId>greeter-lib</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
```

- Was ist der Unterschied zwischen dem Modul `app-main` und dem Modul `greeter-lib`? Das `app-main` kann ausgeführt werden. Während dem `greeter-lib` nur Code ist.

3.2.2 Git Submodule Maven Project

`greeter-lib` in ein eigenes Git-Repo verschieben.

Schritt 1

```
mira@ultramarine multimoduleproject on  main [!] is 📦 v1.0-SNAPSHOT via ☕ v25.0.4 via Ⓜ
❯ cd submodules/greeter-lib && git init && git remote add origin git@git.foobar.vip:school-bbw/fourth-year/modul-324/greeter-lib-subrepo.git
Leeres Git-Repository in /home/mira/Dokumente/git/git.foobar.vip/school-bbw/fourth-year/modul-324/multimoduleproject/submodules/greeter-lib/.git/ initialisiert

mira@ultramarine greeter-lib on  main [?] via ☕ v25.0.4 via Ⓜ
❯ vim .gitignore

mira@ultramarine greeter-lib on  main [?] via ☕ v25.0.4 via Ⓜ took 17s
❯ git add . && git status --porcelain=v2
1 A. N... 000000 100644 100644 0000000000000000000000000000000000000000 e09fffd64cba592edb75c72f617d290f9484517a .gitignore
1 A. N... 000000 100644 100644 0000000000000000000000000000000000000000 4b4543a63b0dbfa0a367ad3753254ca38278c8f1 pom.xml
1 A. N... 000000 100644 100644 0000000000000000000000000000000000000000 d8f7ee9865eeedfb58d78df76c0647515b88cd7d src/main/java/ch/bbw/skm/multimodulemaven/greeter/Greeter.java

mira@ultramarine greeter-lib on  main [+] via ☕ v25.0.4 via Ⓜ
❯ git commit -m "init" && git push
[main (Root-Commit) 6eddae0] init
 3 files changed, 162 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 pom.xml
 create mode 100644 src/main/java/ch/bbw/skm/multimodulemaven/greeter/Greeter.java
Objekte aufzählen: 13, fertig.
Zähle Objekte: 100% (13/13), fertig.
Delta-Kompression verwendet bis zu 16 Threads.
Komprimiere Objekte: 100% (5/5), fertig.
Schreibe Objekte: 100% (13/13), 2.30 KiB | 2.30 MiB/s, fertig.
Gesamt 13 (Delta 0), Wiederverwendet 0 (Delta 0), Paket wiederverwendet 0 (von 0)
remote:
remote:
remote: The private project school-bbw/fourth-year/modul-324/greeter-lib-subrepo was successfully created.
remote:
remote: To configure the remote, run:
remote:   git remote add origin git@git.foobar.vip:school-bbw/fourth-year/modul-324/greeter-lib-subrepo.git
remote:
remote: To view the project, visit:
remote:   https://git.foobar.vip/school-bbw/fourth-year/modul-324/greeter-lib-subrepo
remote:
remote:
remote:
To git.foobar.vip:school-bbw/fourth-year/modul-324/greeter-lib-subrepo.git
 * [new branch]      main → main
Branch 'main' folgt nun 'origin/main'.
```

Schritt 2

```
mira@ultramarine multimoduleproject on  main [!?] is 📦 v1.0-SNAPSHOT via ☕ v25.0.4 via Ⓜ
❯ git submodule add git@git.foobar.vip:school-bbw/fourth-year/modul-324/greeter-lib-subrepo.git greeter-lib
Klone nach '/home/mira/Dokumente/git/git.foobar.vip/school-bbw/fourth-year/modul-324/multimoduleproject/greeter-lib'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 13 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Empfange Objekte: 100% (13/13), fertig.
```

> [!WARNING] Darauf achten, dass die Dateien des `greeter-lib` Moduls danach **nicht mehr doppelt** im Hauptprojekt-Repository getrackt sind (z.Bsp. noch unter `submodules/greeter-lib`). Falls der Ordner vorher schon eingecheckte Dateien enthielt, müssen diese mit `git rm` aus dem Hauptrepo entfernt werden, sonst existiert der Code parallel im Submodule und im Hauptrepo.
> [!WARNING] Git Submodule werden beim normalen `git clone` **nicht automatisch mitgeklont** — der `greeter-lib` Ordner bleibt dann leer. Nach dem Clonen zusätzlich ausführen: `git submodule update --init --recursive` (oder direkt beim Clonen: `git clone --recurse-submodules <url>`)

Fragen zum Verständnis

- Was ist ein Git-Submodule? Ein git submodule ist ein Git Objekt welches von git gemanaged wird um Source Code zu externalisieren.
- Wie erstellt man ein Git-Submodule?

```
git submodule add Git-Repo dir-name
```

- In welcher Datei werden die Git-Submodule verwaltet? In `.gitmodules`
- Wie clont man das Hauptprojekt mit all seinen Submodulen?

```
git clone --recursive Git-Repo
```

- Wie speichert man eine im Submodule 'greeter-lib` geänderte Datei im Git-Repository des Submodules?

```
cd greeter-lib
git add <geänderte Datei>
git commit -m "..."
git push
```

- Wie speichert man eine im Hauptprojekt geänderte Datei im Git-Repository des Hauptprojektes?

```
git add <geänderte Datei>
git commit -m "..."
git push
```

- Wie speichert man eine im Submodule 'app-main` geänderte Datei im Git-Repository des Hauptprojektes?

```
cd app-main
git add <geänderte Datei>
git commit -m "..."
git push
```

Oder

```
git add app-main/
git commit -m "..."
git push
```

- Wie speichert das Hauptprojekt die Referenz auf die Version des Submodules greeter-lib?
- .gitmodules — legt fest wo (Pfad) und woher (Remote-URL) das Submodule liegt:

```
[submodule "greeter-lib"]
    path = greeter-lib
    url = git@git.foobar.vip:.../greeter-lib-subrepo.git
```

- Gitlink im Git-Tree — für den Ordner greeter-lib legt Git im Hauptrepo einen speziellen Eintrag vom Typ `160000` an, der direkt auf einen bestimmten Commit-Hash im Submodule-Repo zeigt:

```
git ls-tree HEAD greeter-lib
# 160000 commit 6eddae0fc04b91fbd7f11782b32096f6b1fad018  greeter-lib
```

- Wie aktualisiert man diese Referenz im Hauptprojekt?
- Im Submodule den gewünschten Stand auschecken/pullen (z.Bsp. neuesten Commit vom Remote holen):

```
cd greeter-lib
git pull origin main
cd ..
```

(oder, für alle Submodule aufs Mal, vom Hauptprojekt aus: `git submodule update --remote greeter-lib`)

- Den neuen Stand des Submodules im Hauptprojekt-Commit festhalten — Git erkennt, dass sich der Gitlink geändert hat, sobald der Submodule-HEAD nicht mehr mit dem gespeicherten Commit-Hash übereinstimmt (`git status` zeigt dann `modified: greeter-lib (new commits)`):

```
git add greeter-lib
git commit -m "update greeter-lib submodule reference"
git push
```

3.2 IDE-Ansatz vs. Maven/Gradle

IDE-Ansatz (IntelliJ Artifacts): IntelliJ hat ein eigenes, Maven-unabhängiges Artefaktmanagement. Unter `File -> Project Structure -> Artifacts` definiert man Name, Typ (z.Bsp. JAR), Output-Verzeichnis und ob das Artefakt beim Project Build automatisch mitgebaut wird; gebaut wird via `Build -> Build Artifacts`. Einbinden eines fremden Artefakts ohne Maven/Gradle: `Project Structure -> Modules -> Dependencies -> + -> JARs or Directories` und die JAR-Datei manuell auswählen. Die Konfiguration landet in den `.idea` / `.iml`-Dateien.
Maven/Gradle-Ansatz: Artefakt wird deklarativ in `pom.xml` (`<dependency>` mit groupId/artifactId/version) bzw. `build.gradle` (`implementation '...'`) eingetragen. Das Tool lädt es automatisch aus einem Repository (Maven Central, lokal `~/.m2`) inkl. aller transitiven Abhängigkeiten. Build via `mvn package` / `gradle build`.

	IDE-Ansatz	Maven/Gradle
➕	Schnell per GUI, kein Setup, gut für kleine Experimente	Portabel (CLI, CI/CD, jede IDE), Versionierung, transitive Dependencies, Repositories
➖	Proprietär (an IntelliJ gebunden), nicht CI/CD-fähig, JARs manuell beschaffen/aktualisieren, keine transitiven Dependencies	Mehr Initialaufwand, XML/DSL lernen

Fazit: IDE-Artefakte sind für lokale Ausnahmen okay; für alles Teamfähige und Automatisierte (DevOps!) sind Maven/Gradle der Standard, weil der Build reproduzierbar und toolunabhängig ist.

3.2 Refresher: Was ist ein Maven-Artefakt?

3.2.3 Bedeutung von Begriffen

- **GroupId:** uniquely identifies a project group across all other groups. Each groupId should follow Java's package name rules. This means it starts with a reversed domain name you control. *Examples:*

```
org.apache.maven
org.apache.commons
com.google.guava
```

- **ArtifactId:** is the name of the artifact. The identifiers should only consist of lowercase letters, digits, and hyphens. *Examples:*

```
commons-math
maven-clean-plugin
```

- **Version:`** follow the rules of Semantic Versioning 1.0.0. It should start with the major version, followed by the minor version and the patch version. All three are numeric, separated by a dot. You can add labels for pre-releases or build metadata after the patch version. Avoid using dates in those labels, because they are usually associated with unstable versions. *Examples:*

```
1.0.0
2.3.2
3.5.42
// Pre-releases
1.0.0-beta
1.0.0-M1
1.0.0-rc2

// Build metadata
1.2.3+dfc0c87
2.3.4+15433
```

- **Packaging:** give us a really complete what: that is the project's packaging (`JAR|WAR`)
- **Classifier:** The classifier distinguishes artifacts that were built from the same POM but differ in content. It is some optional and arbitrary string that - if present - is appended to the artifact name just after the version number. As a motivation for this element, consider for example a project that offers an artifact targeting Java 11 but at the same time also an artifact that still supports Java 1.8. The first artifact could be equipped with the classifier `jdk11` and the second one with `jdk8` such that clients can choose which one to use.

3.2.4 Aufbau eines Maven Artifakt

> Erarbeiten Sie den Aufbau eines Maven-Artefakts.Aufbau festhalten und erklären.

Ein Maven-Artefakt ist eine Datei (meistens ein JAR), die in einem Maven-Repository abgelegt und über ihre **Koordinaten** eindeutig identifiziert wird:

```
groupId:artifactId:packaging:classifier:version
```

Beispiel: `org.apache.commons:commons-lang3:jar:3.12.0`
Nur `groupId:artifactId:version` (**GAV**) sind Pflicht — `packaging` (Default: `jar`) und `classifier` sind optional.
Dateiname nach Konvention:

```
<artifactId>-<version>[-<classifier>].<packaging>
z.Bsp. greeter-lib-1.0-SNAPSHOT.jar
       commons-lang3-3.12.0-sources.jar
```

Ablage im Repository — die Koordinaten bestimmen den Pfad:

```
<repo>/org/apache/commons/commons-lang3/3.12.0/commons-lang3-3.12.0.jar
       └── groupId (Punkte = Ordner) ──┘└ artifactId ┘└ version ┘
```

Inhalt eines JAR-Artefakts: - kompilierte `.class`-Dateien (in der Package-Struktur) - `META-INF/MANIFEST.MF` (Metadaten, ggf. Main-Class) - `META-INF/maven/<groupId>/<artifactId>/pom.xml` + `pom.properties` — das Artefakt trägt seine eigene Baubeschreibung mit
Daneben liegen im Repository zum selben Artefakt typischerweise die `.pom`-Datei (für transitive Dependencies), Checksummen (`.sha1`) und optional `-sources.jar` / `-javadoc.jar` als Classifier-Varianten.

3.2 Lokales Maven-Repository

- Path (Linux/*NIX): `~/.m2`

3.2.5 Custom LocalRepo Config File

Add the following snippet to `~/.m2/settings.xml`:

```
<settings>
    <localRepository>/path/to/maven_repository</localRepository>
    ...
```

3.2.6 Custom LocalRepo CLI

```
mvn -Dmaven.repo.local=/path/to/maven_repository clean install
```

> [!NOTE] Important argument is `-Dmaven.repo.local=/path/to/maven_repository`

3.2 Artefakte zur Verfügung stellen

> Recherchiere Möglichkeiten, ein Maven-Artefakt zu veröffentlichen (Maven Central, Nexus/Artifactory, GitHub Packages, File-Sharing). Beschreiben kurz zu den jeweiligen Möglichkeiten, wie man vorgeht, um ein Artefakt zu veröffentlichen. Halten Sie Vor- und Nachteile der jeweiligen Möglichkeit fest.

3.2.7 GitLab Packages

Vorgehen: In `pom.xml` `<distributionManagement>` mit der GitLab-Registry-URL (`.../api/v4/projects/<id>/packages/maven`) eintragen, Auth-Token in `settings.xml`, dann `mvn deploy`. - ➕ Direkt beim Repo (CI/CD-Integration), private Packages gratis - ➖ Nur für Nutzer mit GitLab-Zugang, Token-Handling nötig

3.2.8 Maven Central

Vorgehen: Account beim Central Portal, groupId (Domain) verifizieren, Artefakt muss Sources + Javadoc + GPG-Signatur enthalten, dann via `maven-publish` / `central-publishing-maven-plugin` deployen. - ➕ Weltweit standardmässig verfügbar, kein Repo-Eintrag beim Konsumenten nötig - ➖ Aufwändiges Setup, strenge Anforderungen, nur öffentlich, kein Löschen möglich

3.2.9 Nexus/Artifactory

Vorgehen: Eigenen Repository-Server betreiben, Repo anlegen, URL in `<distributionManagement>` + Credentials in `settings.xml`, `mvn deploy`. - ➕ Volle Kontrolle, privat, Proxy/Cache für Central - ➖ Server selbst betreiben und warten (Kosten, Aufwand)

3.2.10 File-Sharing

Vorgehen: `mvn package`, JAR (+ pom) manuell verschicken/ablegen (Mail, Ordner, Cloud); Empfänger installiert mit `mvn install:install-file -Dfile=... -DgroupId=... -DartifactId=... -Dversion=...`. - ➕ Null Infrastruktur, sofort - ➖ Keine Versionierung/transitive Dependencies, manuell, fehleranfällig — nicht teamtauglich

---

3.2 Sources

- https://bbw-it.github.io/324_main_rupe/14_MavenArtefacts/14.1-JavaProjektUndArtefakte/
- https://bbw-it.github.io/324_main_rupe/50_KnowledgeBase/14_MavenArtefacts/MultiModuleMavenProject/
- https://bbw-it.github.io/324_main_rupe/50_KnowledgeBase/14_MavenArtefacts/GitSubmoduleMavenProject/
- https://www.jetbrains.com/help/idea/artifacts.html
- https://maven.apache.org/guides/mini/guide-naming-conventions.html
- https://maven.apache.org/pom.html#packaging
- https://www.baeldung.com/maven-local-repository
- https://maven.apache.org/settings.html