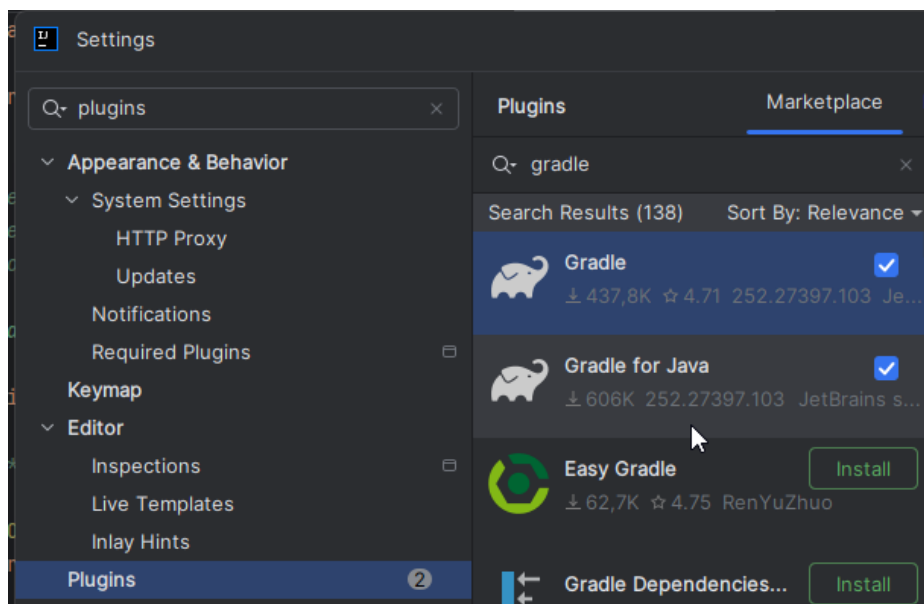


MapReduce job mit Java in IntelliJ IDE

1. Einleitung

IntelliJ community Edition (lizenzfrei) bzw. für Studenten auch die Ultimate Edition (Registrierung mit gültiger FH-Email-Adresse) von <https://www.jetbrains.com/idea/download/?section=windows> herunterladen. Für diese Übung wäre Community Edition ausreichend, für das Studium generell würde ich die Ultimate Edition empfehlen.

Um gradle als Buildsystem zu verwenden, ist es am einfachsten, das plugin für IntelliJ zu installieren. Für die Verwendung auf der Kommandozeile kann es nötig sein, den Installationspfad des gradle-plugins in den Pfad unter „Umgebungsvariablen“ aufzunehmen oder Gradle direkt von <https://gradle.org/releases/> herunterzuladen und zu installieren.



2. Generell

Folgende 2 Varianten bieten sich an, das Projekt zu bauen:

1. Mit Hilfe eines externen build systems (vorbereitetes build.gradle File)
2. Mit Hilfe von IntelliJ selbst als Build-System (erfordert etwas mehr Vorbereitung, ist aber vielleicht besser zu verstehen)

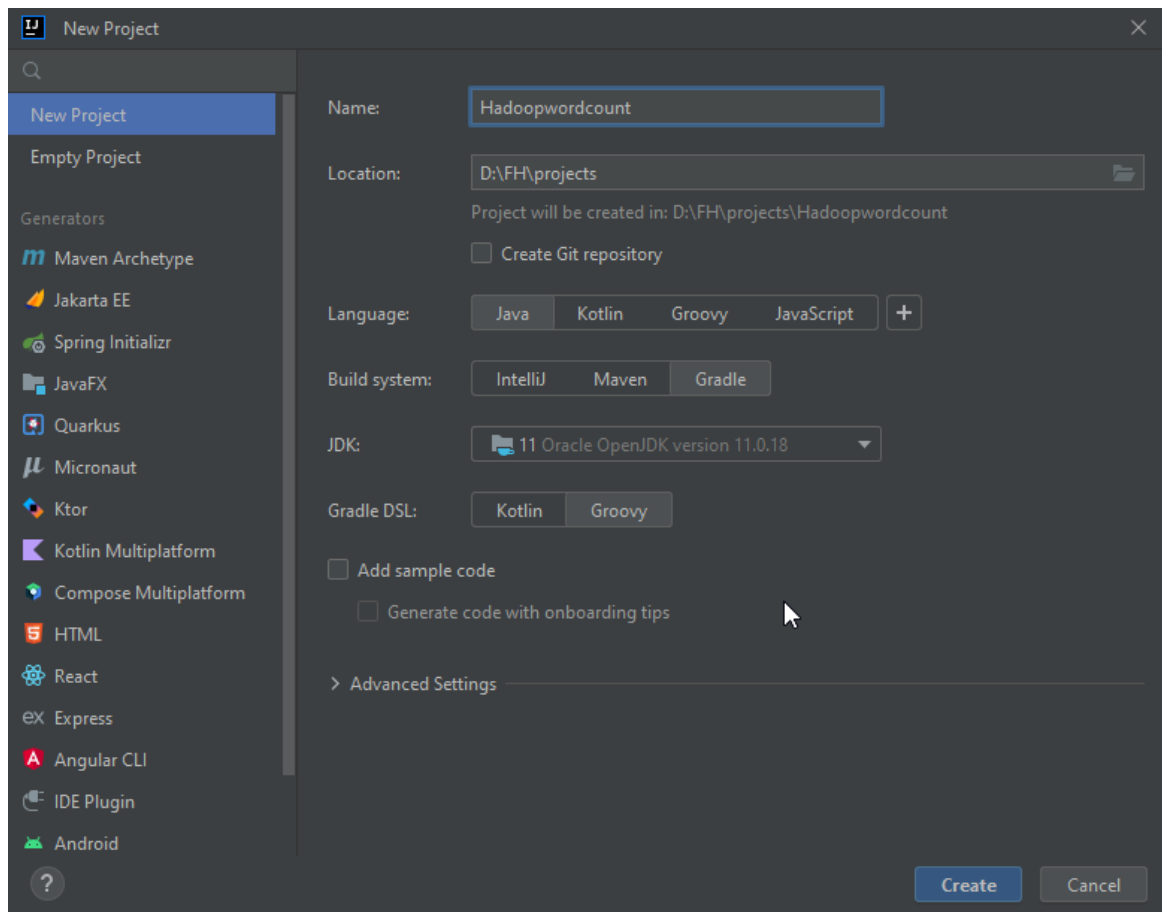
Im Folgenden wird bei Abweichungen zwischen den Varianten stets die Notation @gradle bzw. @intellij verwendet.

3. Projekt anlegen

Nach Installation ein neues Java-Projekt anlegen, passende JDK-Version auswählen.

@gradle: „Gradle“ als Buildsystem auswählen

@intellij: „IntelliJ“ als Buildsystem auswählen



4. Buildsystem Vorbereitungen

@gradle:

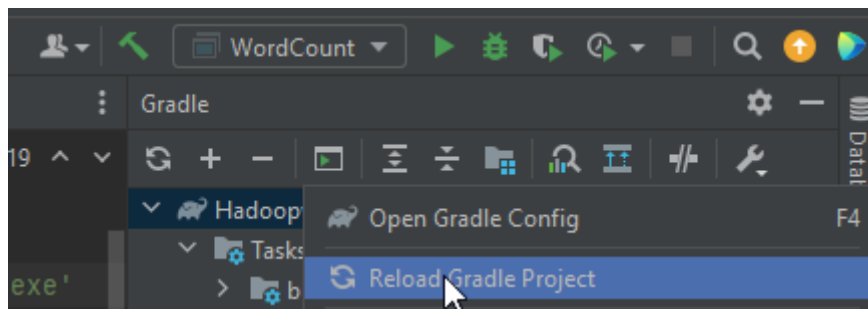
das automatisch generierte build.gradle sollte wie folgt aussehen, in einem ersten Schritt ist nur die "group" ist zu ändern.

```

1  plugins {
2      id 'java'
3  }
4
5  group = 'at.fhj'
6  version = '1.0-SNAPSHOT'
7
8  repositories {
9      mavenCentral()
10 }
11
12 dependencies {
13     testImplementation platform('org.junit:junit-bom:5.9.1')
14     testImplementation 'org.junit.jupiter:junit-jupiter'
15 }
16
17 test {
18     useJUnitPlatform()
19 }

```

Besser jedoch das build.gradle aus dem github-Repo (<https://github.com/Woberegger/BigData/tree/main/src/wordcount>) abholen, dort sind auch die dependencies schon eingetragen – danach folgendes ausführen, damit das build-File neu geladen wird – es sollten darauf hin etliche Jar-Files aus dem zentralen Repo im Internet runtergeladen werden:

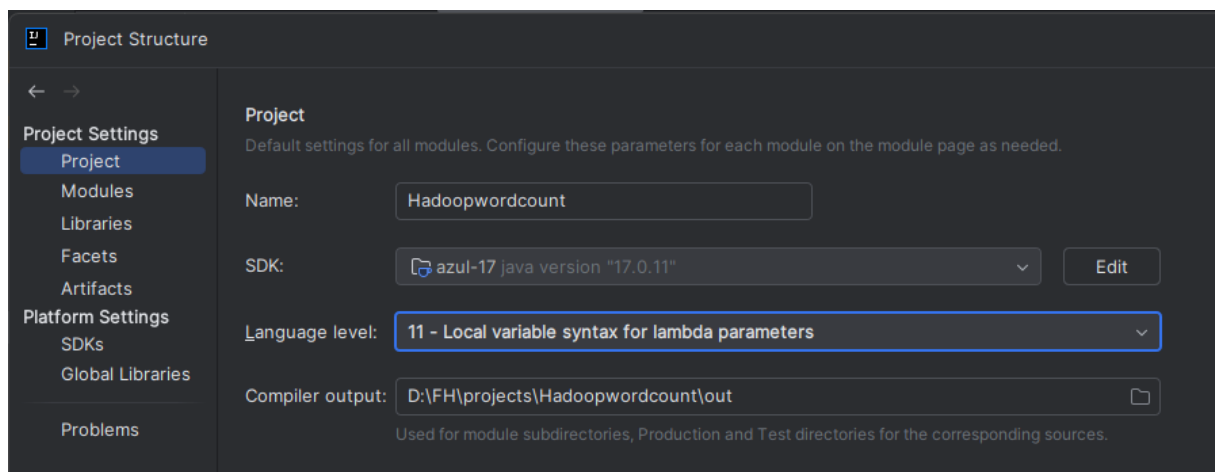


Im build.gradle ist wichtig, dass die Kompatibilität zu älterer Java-Version gesetzt ist:

```
java {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}
```

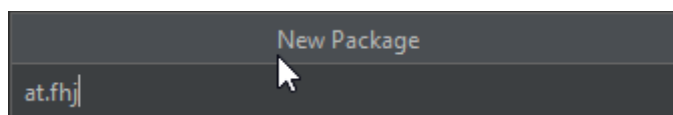
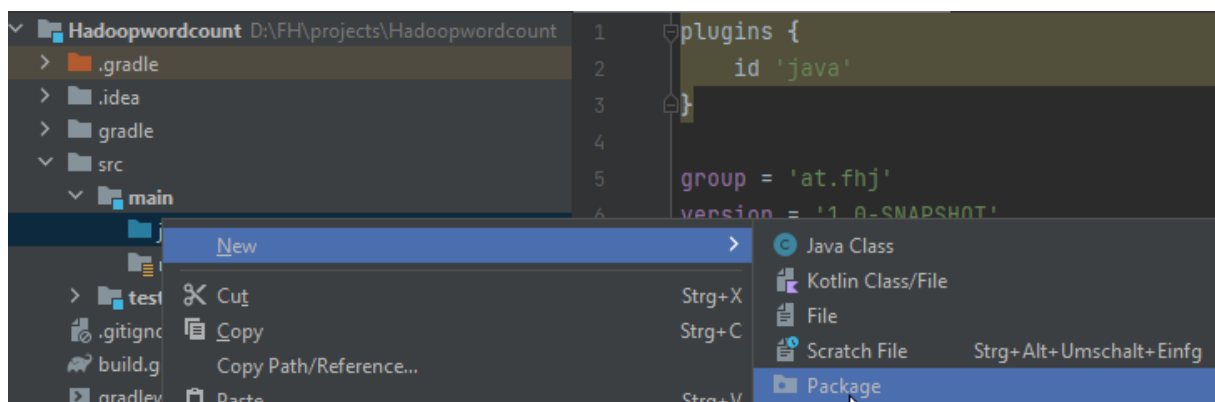
@intellij

Wenn eine neuere Java-Version als 11 verwendet wird, hier die Kompatibilität zu Version 11 eintragen (bei Verwendung von gradle wird dies als „compatibility“ im build.gradle eingetragen):



5. Existierende Sourcefile vorbereiten

ein neues Package at.fhj anlegen, damit die Struktur der Defaultstruktur entspricht.



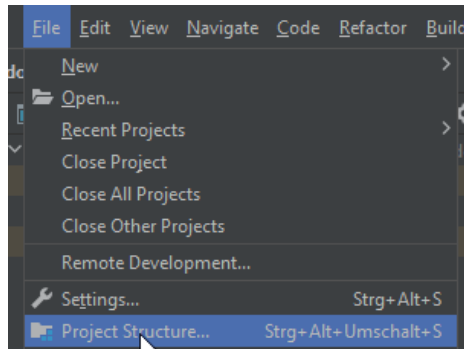
Dort die 3 Sourcefiles *.java ablegen (am einfachsten durch Kopieren von <https://github.com/Woberegger/BigData/tree/main/src/wordcount> in das zuvor angelegte Package-Directory)

6. Projektstruktur anpassen

@gradle

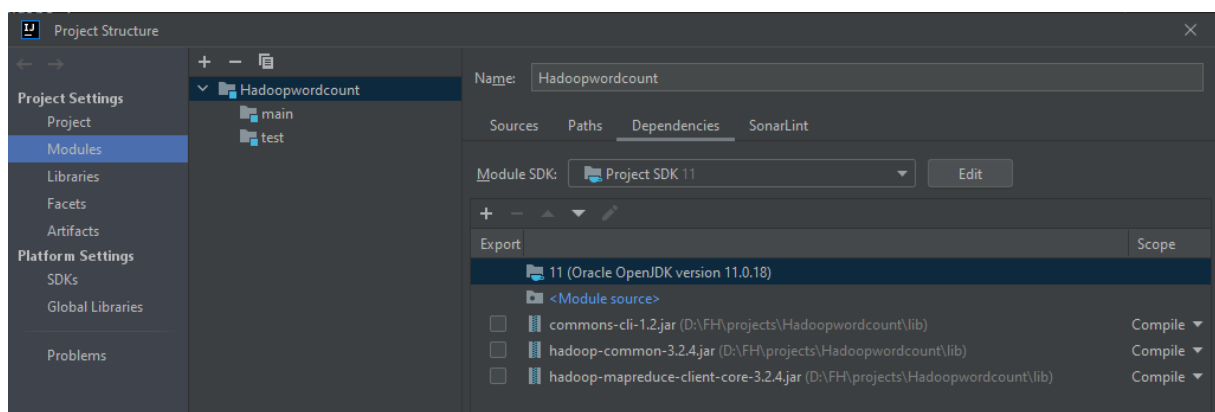
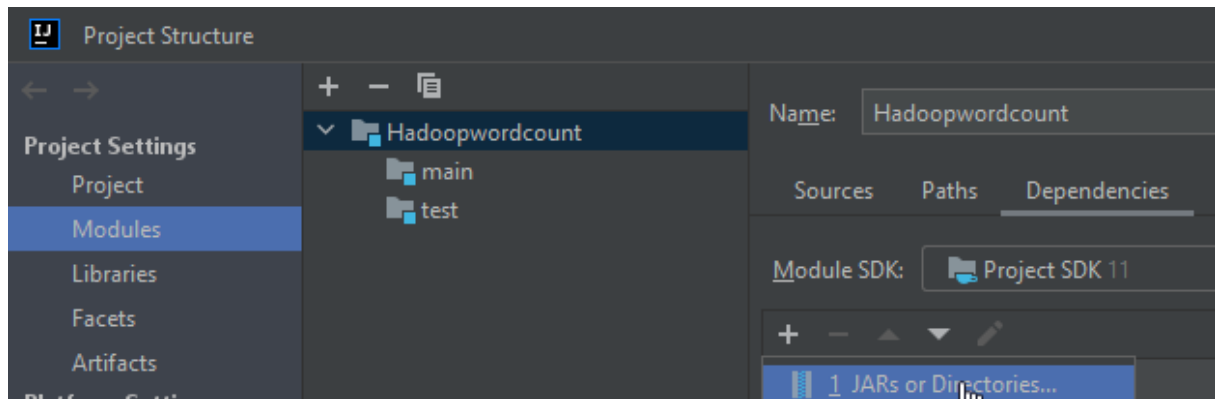
Nichts zu tun

@intellij

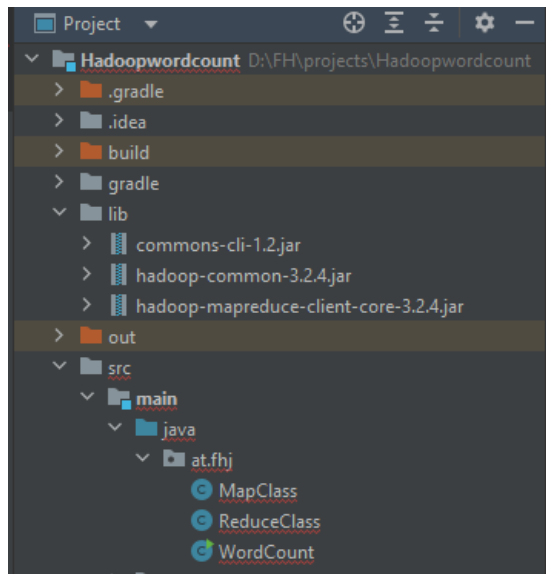


Damit die Dependencies aufgelöst werden, gibt es mehrere Möglichkeiten:

- MavenCentral repo verwenden (siehe build.gradle)
- Jar-Dateien zum Projekt hinzufügen (die Dateien entweder aus der E-Learning-Plattform oder noch besser vom installierten hadoop in der virtuellen Maschine – ACHTUNG: Versionen müssen zusammenpassen, es ist aber möglich, mit niedrigerer Version zu bauen, als die Version in der virtuellen Maschine):



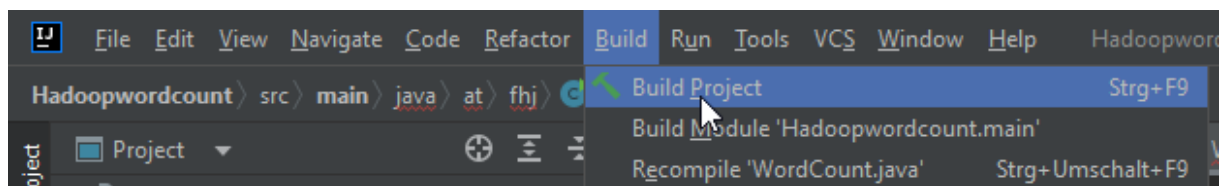
Projektstruktur, nachdem die 3 Sourcedateien (*.java) unter „src/main/java/at/fhj“ und 3 Libraries (*.jar) unter „lib“ abgelegt wurden



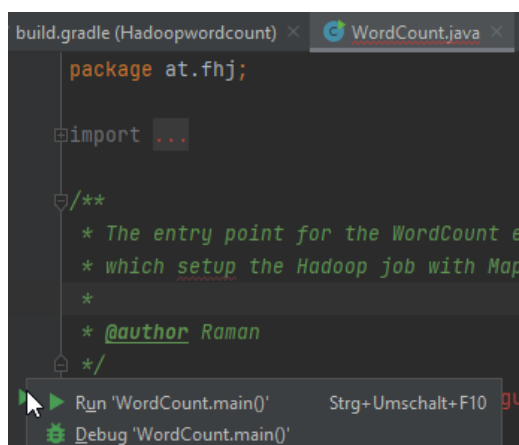
7. Project bauen

8. @intellij

Projekt bauen, d.h. kompilieren



Ausführen oder Debuggen direkt bei den jeweiligen Methoden (müsste man remote debuggen auf der Zielplattform verwenden, wo DFS installiert ist), d.h. dieser Schritt ist im Normalfall zu überspringen.



@gradle

Wie bei IntelliJ bzw. optional über Kommandozeile

```
Terminal Local x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

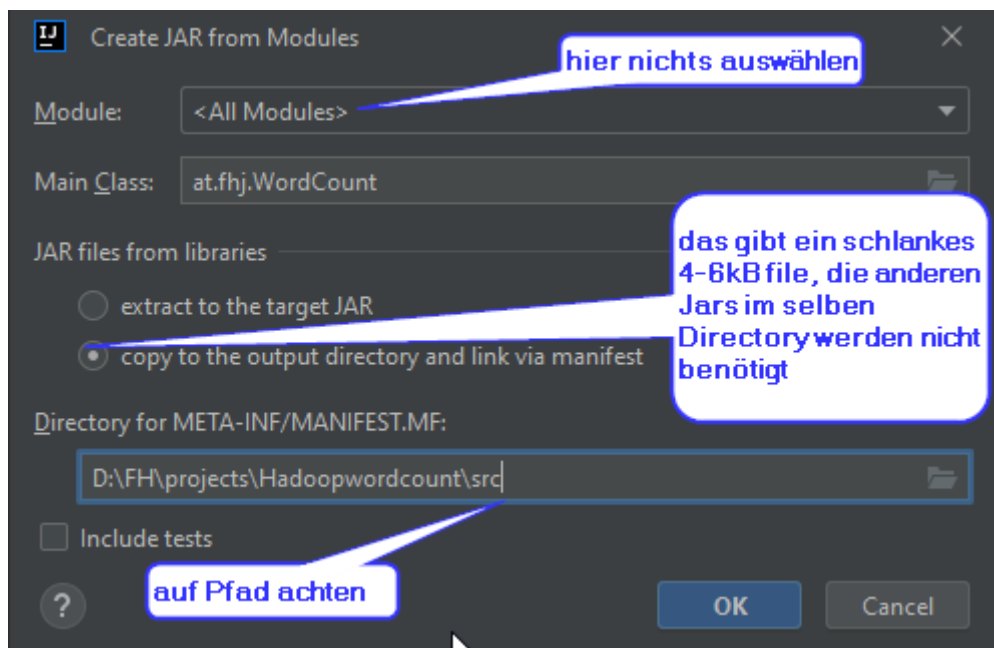
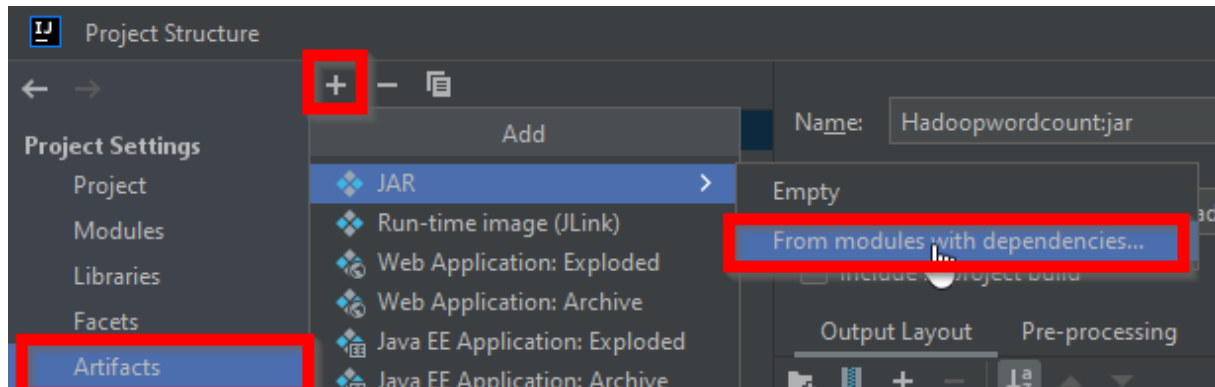
Installieren Sie die neueste PowerShell für neue Funktionen und Verbesserungen! https://aka.ms/PSWindows

PS D:\FH\projects\Hadoopwordcount> gradle build
```

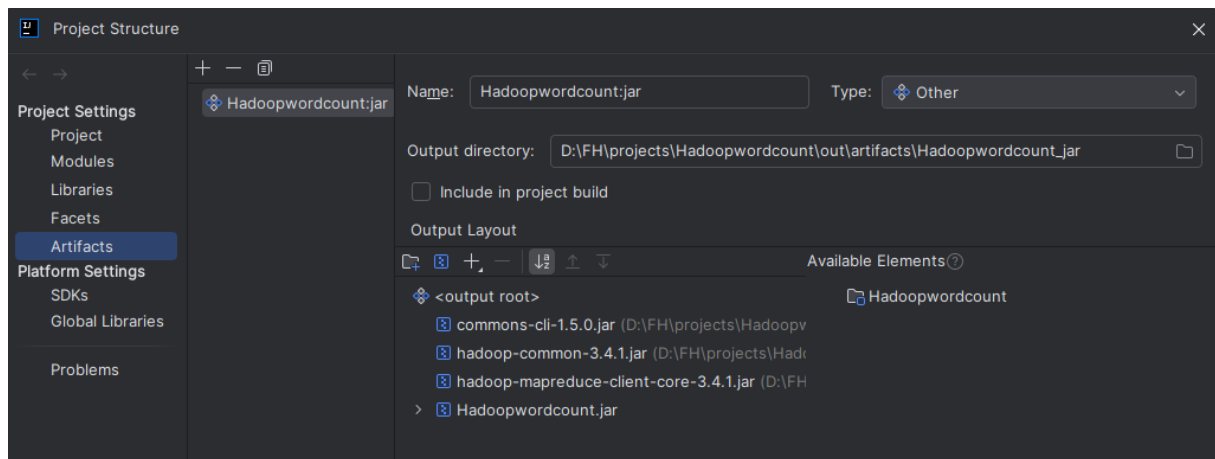
9. Jar-Datei erstellen

@intellij

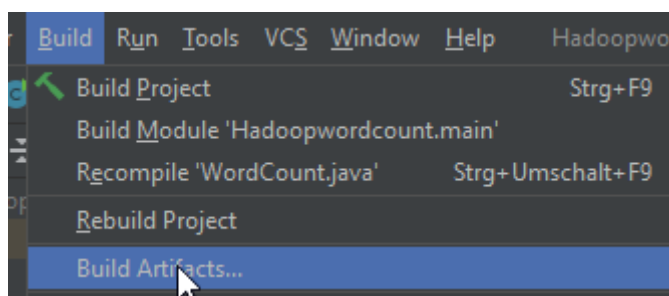
Artifakt anlegen, d.h. über "Project structure" das Erstellen des jar-File konfigurieren und dem Jar-File mitteilen, wo der Einsprungspunkt, d.h. die Main-Klasse, zu finden ist.



erwarteter Output danach ungefähr wie folgt, diesen mit "Apply" bestätigen (wenn mit gradle davor schon ein build angestoßen wurde, kann sein, dass hier die Liste auch bereits länger ist):



und dann kann man das Artifikat bauen:



@gradle

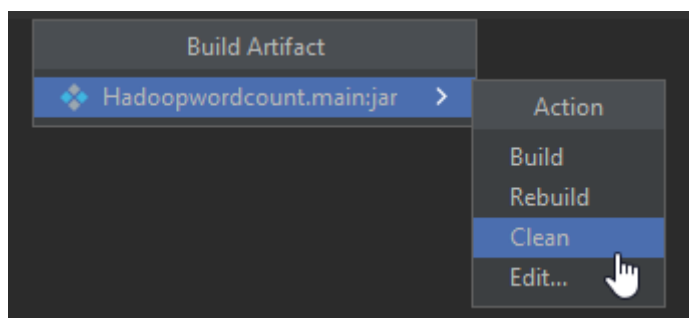
Entweder ebenfalls über die GUI oder über Kommandozeile:

gradle task Jar

10. Prüfung des erstellten Artifacts und Kopieren auf Zielplattform

Je nach verwendetem Buildsystem und deren Settings findet man das generierte Jar-File unter z.B. out/artifacts oder build/libs und beinhaltet der Dateiname eine Versionsnr. oder nicht.

Falls zuvor schon eine Version dort gebildet wurde, die nicht funktioniert (weil z.B. fehlende Dependencies), diese besser über "Clean" zuerst löschen oder noch besser gesamtes "out" Verzeichnis löschen.



Das build.gradle beinhaltet einen „deploy“ Task, dieser funktioniert jedoch nur bei Passwort-based login bzw. müsste man die Credentials und Hostnamen ändern.

Daher am besten händisch die Datei manuell kopieren als user „student“ und dann die Datei dem hduser „schenken“:

```
scp Hadoopwordcount-1.0.jar student@<hostname>:/tmp/Hadoopwordcount-1.0.jar
ssh student@<hostname>
sudo chown hduser:hadoop /tmp/Hadoopwordcount-1.0.jar
su - hduser
mv /tmp/Hadoopwordcount-1.0.jar ~/Hadoopwordcount.jar
```

11. Test des generierten jar-Files

Hadoop starten und Test aufrufen mit dem zuvor generierten und auf die Zielplattform kopierten Jar-File.

Job ausführen

- 1.) Input Ordner und Beispiel Textdatei im hdfs anlegen
- 2.) Job ausführen (Vorsicht Pfade sind ggf. anzupassen!)

```
su - hduser
start-dfs.sh
start-yarn.sh
hdfs dfs -mkdir /input
hdfs dfs -mkdir /output
hdfs dfs -put ~/BigData/data/Bibel.txt /input/
OutputDir=/output/Bibel
# folgendes für jeden Versuch neu ausführen
hdfs dfs -rm -R $OutputDir
hadoop jar Hadoopwordcount.jar /input/Bibel.txt $OutputDir
```

Bei Problemen (vor allem wenn Meldung „main class not found“) die Jar-Datei überprüfen ("jar -tvf" funktioniert meines Wissens nur unter Linux), sie muss zumindest folgende 4 Dateien beinhalten:

```
jar -tvf Hadoopwordcount.jar | awk '{ print $8 }'
META-INF/MANIFEST.MF
at/fhj/MapClass.class
at/fhj/ReduceClass.class
at/fhj/WordCount.class
```

Der Inhalt der Datei MANIFEST.MF sollte wie folgt sein (optional stehen dahinter noch ClassPaths):

```
jar -xf Hadoopwordcount.jar META-INF/MANIFEST.MF
cat META-INF/MANIFEST.MF
Manifest-Version: 1.0
Main-Class: at.fhj.WordCount
```

Erwarteter Output nach erfolgreichem MapReduce Job wie folgt:


```
hduser@osboxes: ~/Desktop

Reduce input groups=56030
Reduce shuffle bytes=9199249
Reduce input records=781853
Reduce output records=56030
Spilled Records=1563706
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=55
Total committed heap usage (bytes)=706740224

Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0

File Input Format Counters
  Bytes Read=4579278
File Output Format Counters
  Bytes Written=614705

Job was successful
hduser@osboxes:~/Desktop$
```

Im angegebenen Output-Verzeichnis befinden sich dann 2 Dateien, eine leere Datei "_SUCCESS" und eine Datei mit dem Ergebnis des Jobs (wäre der Output größer als konfigurierte Blockgröße würde es weitere Dateien part-r-00001 usw. geben).

localhost:9870/explorer.html#/output/bibel 110%

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

/output/bibel Go! [Icons]

Show 25 entries Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	hduser	supergroup	0 B	Nov 04 15:18	2	16 MB	_SUCCESS	
<input type="checkbox"/>	-rw-r--r--	hduser	supergroup	363.59 KB	Nov 04 15:18	2	16 MB	part-r-00000	

File information - part-r-00000

Download
Head the file (first 32K)
Tail the file (last 32K)

Block information --
Block 0

Block ID: 1073741956
Block Pool ID: BP-109114966-127.0.1.1-1698821495461
Generation Stamp: 1138
Size: 372320
Availability:

- UbuntuBigData

File contents

```

mme      1
verstummen  1
verstummt  2
verstummt  1

```

Das "Tail the file" funktioniert üblicherweise nicht über die GUI, daher besser über Kommandozeile:


```
hdfs dfs -tail /output/Bibel/part-r-00000
hdfs dfs -head /output/Bibel/part-r-00000
```

Auf der Webseite von Yarn sieht man die erledigten Jobs ebenfalls (zu sehen nur im Fall, wenn mapreduce.framework.name auf „yarn“ und nicht „local“ eingestellt wurde):

← → ↺

Nicht sicher namenode:8088/cluster/apps/FINISHED

🔍 ☆ ⌂ ⌵



Cluster

About
Nodes
Node Labels
Applications
NEW
NEW SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running
1	0	1	0	<1

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned
3	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime
application_1763115893210_0003	hduser	my word count	MAPREDUCE		root.default	0	Fri Nov 14 11:31:47 +0100 2025	Fri Nov 14 11:31:49 +0100 2025

Showing 1 to 1 of 1 entries

Wichtig: wenn der Job erneut gestartet werden soll, muss zuvor das Output-Verzeichnis gelöscht werden (sowohl wenn man Output ins lokale Dateisystem als auch wenn man dies in hdfs schreibt)!