

JIGSAW - Cheat Sheet

Copyright Michael Inden, 2016

Hinweise:

- **Module** stellen eine **weitere Hierarchie zu Packages** dar
- **IDEs** können das **noch nicht** abbilden
- Java-Compiler und die JVM wurde entsprechend erweitert:
 - `javac --module-source-path src -d build $(find src -name '*.java')`
 - `java --module-path <modulepath> -m <modulename>/<moduleclass>`

ACHTUNG: Noch wildes Durcheinander im JDK `--module-path` und `-modulepath` usw.
=> auch in den Ausgaben der Kommandozeilen-Tools!

1) Hauptverzeichnis anlegen

```
mkdir jdk9workshop
cd jdk9workshop
```

***optional: tree installieren:** `brew install tree`

2) Modul-Verzeichnis im src-Verzeichnis anlegen

```
mkdir -p src/myfirstmodule
```

3) Moduldeskriptor = Module-Info-Datei anlegen

```
cat > src/myfirstmodule/module-info.java
module myfirstmodule
{
}
```

***CTRL-C**

4) Verzeichnis- bzw. Package-Hierarchie anlegen

```
mkdir -p src/myfirstmodule/com/hellojigsaw
```

5) Applikationsklasse erstellen

```
> cat > src/myfirstmodule/com/hellojigsaw/HelloJigsaw.java
package com.hellojigsaw;
```

```
public class HelloJigsaw
{
    public static void main(final String[] args)
    {
        System.out.println("Hello Jigsaw!");
    }
}
```

6) Verzeichnisstruktur und Inhalt prüfen

```
> tree
```

```
└─ src
   └─ myfirstmodule
      ├── com
      │   └─ hellojigsaw
      │       └─ HelloJigsaw.java
      └─ module-info.java
```

7) Klasse als Modul kompilieren

```
> javac -d build/myfirstmodule \
    src/myfirstmodule/module-info.java \
    src/myfirstmodule/com/hellojigsaw/HelloJigsaw.java
```

8) Verzeichnisstruktur und Inhalt prüfen

```
> tree
```

```
└─ .
   ├── build
   │   └─ myfirstmodule
   │       ├── com
   │       │   └─ hellojigsaw
   │       │       └─ HelloJigsaw.class
   │       └─ module-info.class
   └─ src
       └─ myfirstmodule
           ├── com
           │   └─ hellojigsaw
           │       └─ HelloJigsaw.java
           └─ module-info.java
```

9) Starten des Programms aus einem Modul

```
> java --module-path build -m myfirstmodule/com.hellojigsaw.HelloJigsaw
```

Parameter:

--module-path => Pfad zu den Modulen (Kurzform -p)

-m => Modul & Start-Klasse

10) Deployable JAR erstellen

```
> mkdir lib
```

```
> jar --create --file lib/myfirstmodule_1.0.jar --module-version 1.0 -C
build/myfirstmodule .
```

=>

```
|—— lib
|   |—— myfirstmodule_1.0.jar
```

Parameter:

--create => Archiv erzeugen (hier lieber --create, statt Kurzform -c)

--file => Archivfile

--module-version => Versionsnummer des Moduls

-C => Dateien aus dem Verzeichnis nutzen

=> Ein Archiv ist ein modulares JAR-Archiv, wenn der Moduledeskriptor "module-info.class" in der Root der angegebenen Verzeichnisse oder in der Root des JAR-Archivs selbst vorhanden ist.

11) Abhängigkeiten ermitteln oder prüfen

```
> jdeps lib/*.jar
```

myfirstmodule

```
[file:///Users/michaeli/jdk9workshop/lib/myfirstmodule_1.0.jar]
  requires mandated java.base
```

myfirstmodule -> java.base

com.hellojigsaw	-> java.io	java.base
com.hellojigsaw	-> java.lang	java.base

12) Grafische Aufbereitung

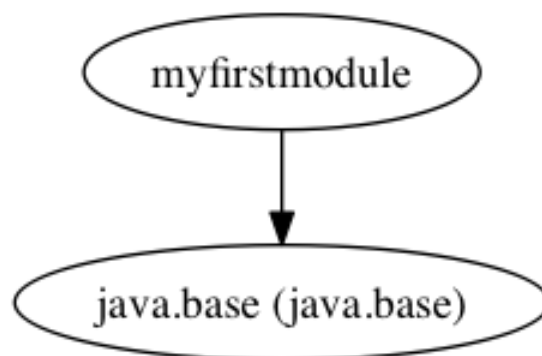
AKTION: GRAPHVIZ installieren (Leider momentan noch nicht mit macOS Sierra)
Windows: http://www.graphviz.org/Download_windows.php
Ubuntu: `sudo apt-get install graphviz`
Mac: `brew install graphviz`

```
> jdeps -dotoutput graphs lib/*.jar
```

```
=>
```

```
└─ graphs
  │   └─ myfirstmodule.dot
  │   └─ summary.dot
```

```
> dot -Tpng graphs/summary.dot > summary.png
> open summary.png
```



13) Ausführbares Modul erzeugen

```
> rm lib/myfirstmodule_1.0.jar
```

```
> jar --create --file lib/myfirstmodule_1.0.jar --main-class
com.hellojigsaw.HelloJigsaw --module-version 1.0 -C build/myfirstmodule .
```

Nun können wir das Programm mit folgendem Kommando starten:

```
> java -p lib -m myfirstmodule
Hello Jigsaw!
```

14) Ausführbares Executable (Runtime Image) erzeugen

```
> jlink --module-path $JAVA_HOME/jmods:lib --add-modules myfirstmodule
--output executablemoduleexample
```

Parameter:

```
--output => Ausgabeverzeichnis
--add-modules => zu inkludierende Module
--module-path => bestimmt den Modulpfad
```

```
> tree executablemoduleexample/
executablemoduleexample/
├── bin
│   ├── myfirstmodule
│   ├── java
│   └── keytool
├── conf
│   ├── net.properties
│   └── security
│       ├── java.policy
│       └── java.security
├── lib
│   ├── jli
│   │   └── libjli.dylib
│   ├── jspawnhelper
│   └── jvm.cfg
...
│   ├── libverify.dylib
│   ├── libzip.dylib
│   ├── modules
│   ├── security
│   │   ├── blacklist
│   │   ├── blacklisted.certs
│   │   ├── cacerts
│   │   ├── default.policy
│   │   └── trusted.libraries
│   ├── server
│   │   ├── Xusage.txt
│   │   ├── libjsig.dylib
│   │   └── libjvm.dylib
│   └── tzdb.dat
└── release
```

```
> executablemoduleexample/bin/myfirstmodule
Hello Jigsaw!
```