JIGSAW - Cheat Sheet V2

KONSOLENVARIANTE

Hinweise:

- ***Module*** stellen eine ***weitere Hierarchie zu Packages*** dar

- 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>`

1) Tools installieren

HOMEBREW: `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`

TREE: `brew install tree`

2) Hauptverzeichnis anlegen

```
> mkdir jigsaw-workshop
> cd jigsaw-workshop
```

3) Modul-Verzeichnis im src-Verzeichnis anlegen

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

4) Moduldeskriptor = Module-Info-Datei anlegen

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

*CTRL-C
```

5) Verzeichnis- bzw. Package-Hierarchie anlegen

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

6) 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!");
    }
}
```

7) Verzeichnisstruktur und Inhalt prüfen

```
> tree

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

8) Klasse als Modul kompilieren

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

9) Verzeichnisstruktur und Inhalt prüfen

```
> tree

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

10) 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
```

11) Deployable JAR erstellen

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

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 Moduldeskriptor "module-info.class" in der Root der angegebenen Verzeichnisse oder in der Root des JAR-Archivs selbst vorhanden ist.

12) Verzeichnisstruktur und Inhalt prüfen

```
├── lib
|     └── myfirstmodule_1.0.jar
```

13) 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
```

```
> jdeps -s lib/*.jar
=>
myfirstmodule -> java.base
```

14) Grafische Aufbereitung

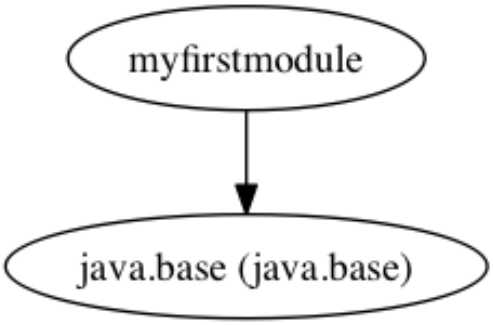
AKTION: GRAPHVIZ installieren
Windows: http://www.graphviz.org/Download_windows.php
Ubuntu: sudo apt-get install graphviz
Mac: brew install graphviz

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

=>

├── graphs
|   ├── myfirstmodule.dot
|   └── summary.dot


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



15) Ausführbares Modul erzeugen

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

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

Nun können wir das Programm mit folgendem Kommando starten:

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

16) Ausführbares Executable (Runtime Image) erzeugen

> /usr/libexec/java_home —verbose

export JAVA_HOME="/Library/Java/JavaVirtualMachines/jdk-9.0.4.jdk/Contents/Home"
export PATH="$JAVA_HOME/bin:$PATH"

```
> jlink --module-path $JAVA_HOME/jmods:lib --add-modules myfirstmodule --
launcher jigsawapp=myfirstmodule/com.hellojigsaw.HelloJigsaw --output
executablemoduleexample
```

Parameter:

```
--module-path => bestimmt den Modulpfad
--add-modules => zu inkludierende Module
--launcher => Angabe der ausführbaren Klasse
--output => Ausgabeverzeichnis
```

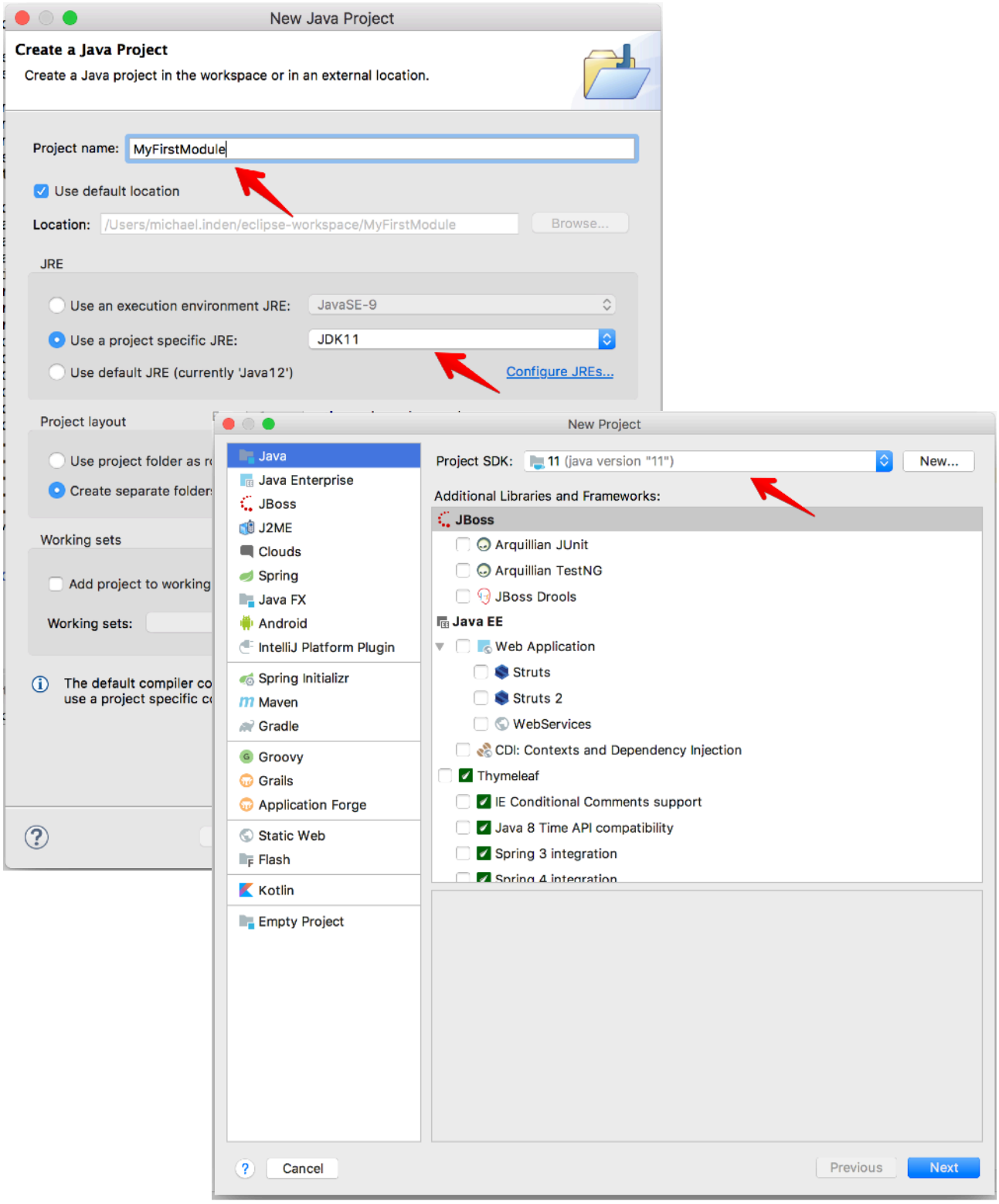
```
> tree executablemoduleexample/
executablemoduleexample/
├── bin
|   ├── jigsawapp
|   ├── java
|   └── keytool
├── conf
|   ├── net.properties
|   └── security
|       ├── java.policy
|       └── java.security
├── lib
|   ├── jli
|   |   └── libjli.dylib
|   ├── jspawnhelper
|   ├── jvm.cfg
...
|   ├── libverify.dylib
|   ├── libzip.dylib
|   ├── modules
|   ├── security
|   |   ├── blacklist
…
|   ├── server
|   |   ├── Xusage.txt
|   |   ├── libjsig.dylib
|   |   └── libjvm.dylib
|   └── tzdb.dat
└── release
```

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

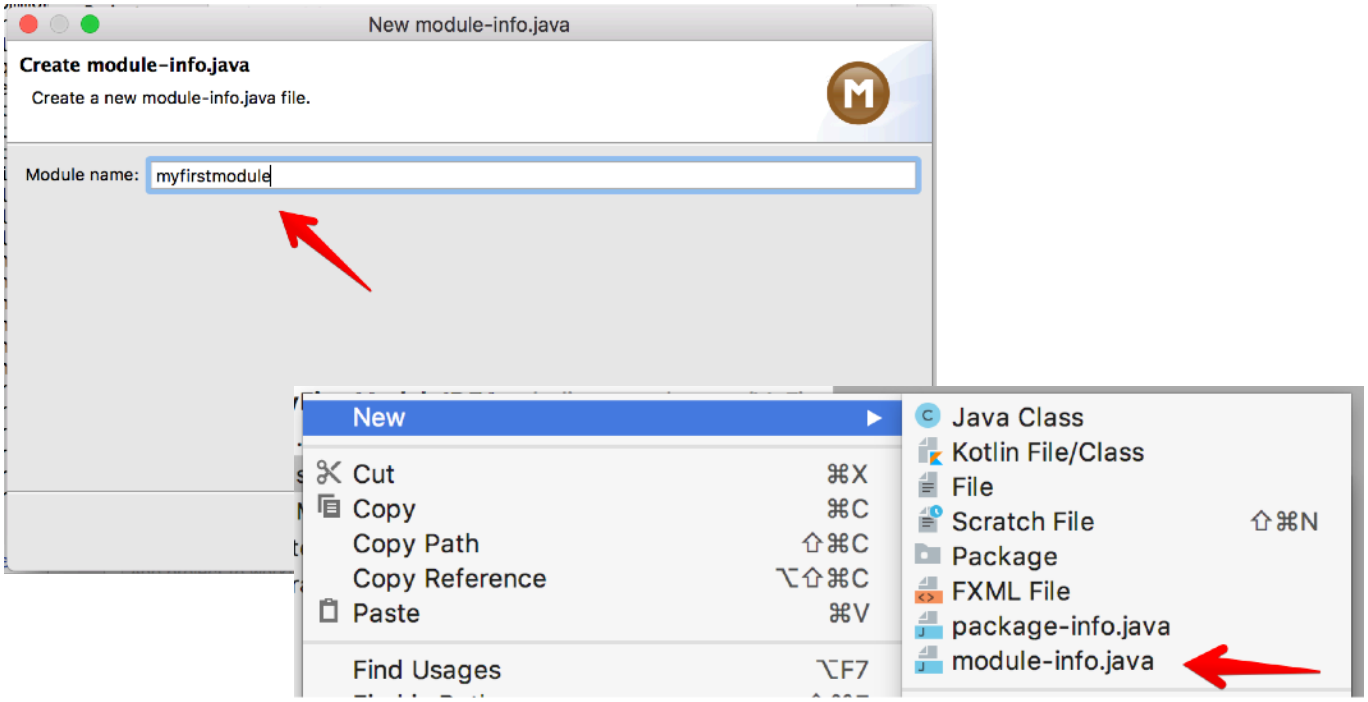
IDE-Version

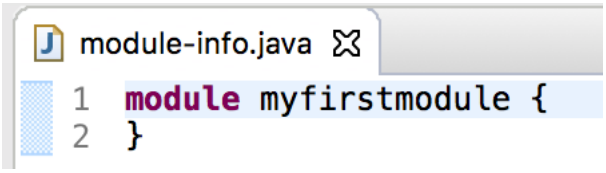
Nutze eine aktuelle IDE deiner Wahl. Nachfolgend ist links Eclipse und rechts nach unten versetzt IntelliJ abgebildet.

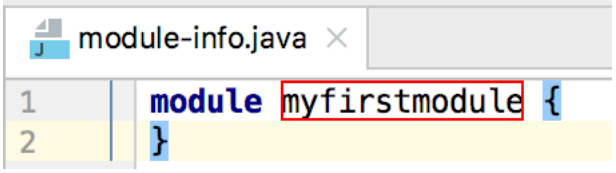
1) Java-Projekt anlegen (New > Java Project / New > Project)



2) Module Deskriptor erstellen







3) Verzeichnis- bzw. Package-Hierarchie anlegen

New > Package com.hellojigsaw

4) Applikationsklasse anlegen

New > Class bzw. New > Java Class HelloJigsaw

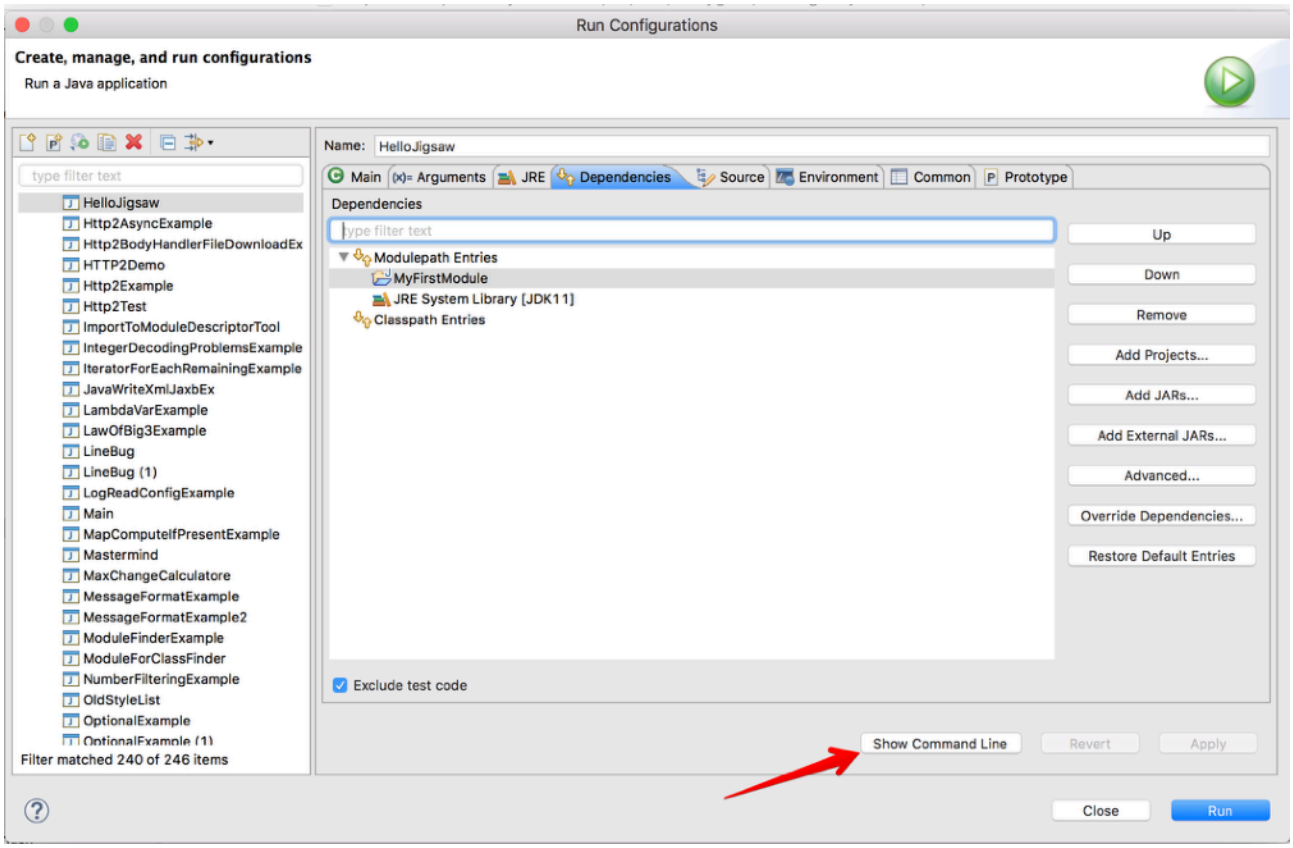
```
package com.hellojigsaw;

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

5) Starten der Applikation als Modul

Starte das Programm wie gewohnt aus der IDE. Analysiere dann Aufruparameter -p … -m …

a) Für IntelliJ ist das einfach, weil diese bei Start angegeben werden.
b) Für Eclipse gibt es folgenden Trick:



Wechsel in das Projektverzeichnis und versuche den Applikationsstart selbst in etwa wie folgt:

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

```
java -p out/production -m myfirstmodule/com.hellojigsaw.HelloJigsaw
```

Zum Ermitteln der Ausgabeverzeichnisse hilft das `tree` Kommando

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

```
├── out
│   └── production
│       └── MyFirstModuleIDEA
│           ├── com
│           │   └── hellojigsaw
│           │       └── HelloJigsaw.class
│           └── module-info.class
└── src
    ├── com
    │   └── hellojigsaw
    │       └── HelloJigsaw.java
    └── module-info.java
```

6) Deployable JAR erstellen

Beachten Sie bitte, dass wir mit IDEs keine „künstliche“ Ebene mit Modulverzeichnissen nutzen, wie dies leider immer noch von Oracle vorgeschlagen wird …

```
> mkdir lib
> jar --create --file lib/myfirstmodule.jar -C bin .
```

Parameter:

```
--create => Archiv erzeugen (hier lieber --create, statt Kurzform -c)
--file => Archivfile
-C => Dateien aus dem Verzeichnis nutzen
```

```
├── lib
│   └── myfirstmodule.jar
```

7) Abhängigkeiten prüfen

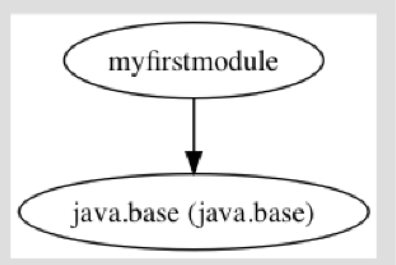
> jdeps lib/*.jar

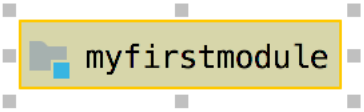
```
jdeps lib/*.jar
myfirstmodule
 [file:///Users/michael.inden/eclipse-workspace/MyFirstModule/lib/
myfirstmodule.jar]
   requires mandated java.base
myfirstmodule -> java.base
   com.hellojigsaw                                        -> java.io
java.base
   com.hellojigsaw                                        -> java.lang
java.base
```

8) Grafische Aufbereitung

AKTION: GRAPHVIZ installieren, wie zuvor im Konsolenteil beschrieben

```
> jdeps -dotoutput graphs lib/*.jar
> dot -Tpng graphs/summary.dot > summary.png
> open summary.png
```





9) Ausführbares Executable (Runtime Image) erzeugen

Aktuelles JDK prüfen

```
> /usr/libexec/java_home –verbose
```

Umgebungsvariable JAVA_HOME prüfen

```
> echo $JAVA_HOME
```

Beispielsweise auf Version 11.0.2 setzen:

```
>  export JAVA_HOME="/Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/
Contents/Home"
> export PATH="$JAVA_HOME/bin:$PATH"
```

```
> jlink --module-path $JAVA_HOME/jmods:lib --add-modules myfirstmodule \
–launcher jigsawapp=myfirstmodule/com.hellojigsaw.HelloJigsaw \
––output executablemoduleexample
```

Alternativ

```
> jlink --module-path $JAVA_HOME/jmods:lib --add-modules myfirstmodule \
–launcher=jigsawapp=myfirstmodule/com.hellojigsaw.HelloJigsaw \
––output executablemoduleexample
```

Parameter:

```
--module-path => bestimmt den Modulpfad
--add-modules => zu inkludierende Module
--launcher => Angabe der ausführbaren Klasse
--output => Ausgabeverzeichnis
```

```
> ./executablemoduleexample/bin/jigsawapp
Hello Jigsaw!
```