# JIGSAW - Cheat Sheet V2

## Console Version

### Clues:

- **Modules represent an additional hierarchy for packages.**

- Java Compiler and JVM were extended accordingly:
  - `javac --module-source-path src -d build $(find src -name '*.java')`
  - `java  --module-path <modulepath> -m <modulename>/<moduleclass>`

### 1) Installation of Tools

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

**TREE**: `brew install tree`

### 2) Creation of main directory

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

### 3) Creation of module dir inside of src dir

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

### 4) Creation of modul descriptor = Module-Info-File

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

*CTRL-C
```

## 5) Creation of directory and package hierarchy

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

## 6) Creation of application class

```
> 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) Check of dir and contents

```
> tree

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

## 8) Compile class as a module

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

## 9) Check of dir and contents

```
> tree

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

## 10) Start of the application within a module

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

**Parameters:**
```
--module-path => path to modules (shortcut -p)
-m => module name and class to execute
```

## 11) Creation of deployable JAR

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

**Parameters:**
```
--create => creation of archive (use –create here to avoid
misunderstandings wit short -c / -C)
--file => archive file
--module-version => version number of the module
-C => include files form the given directory
```

=> An archive is a modular JAR archive if the module descriptor module-info.class is present in the root of the JAR archive itself.

## 12) Check of dir and contents

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

## 13) Check of dependencies

```
> 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) Graphical representation

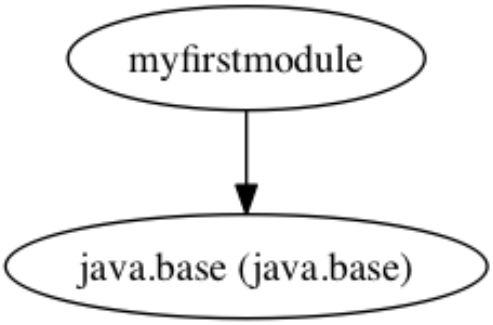
**ACTION: Installation of GRAPHVIZ**
Windows: http://www.graphviz.org/Download_windows.php
Ubuntu: sudo apt-get install graphviz
Mac: brew install graphviz

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

=>

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


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




## 15) Creation of executable module

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

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

Now, we should be able to start the program with the following command:

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

# 16) Creation of Executable (Runtime Image)

**>** /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
```

**Parameters:**

```
--module-path => path to modules
--add-modules => modules to include
--launcher => name of the class to execute
--output => output dir
```

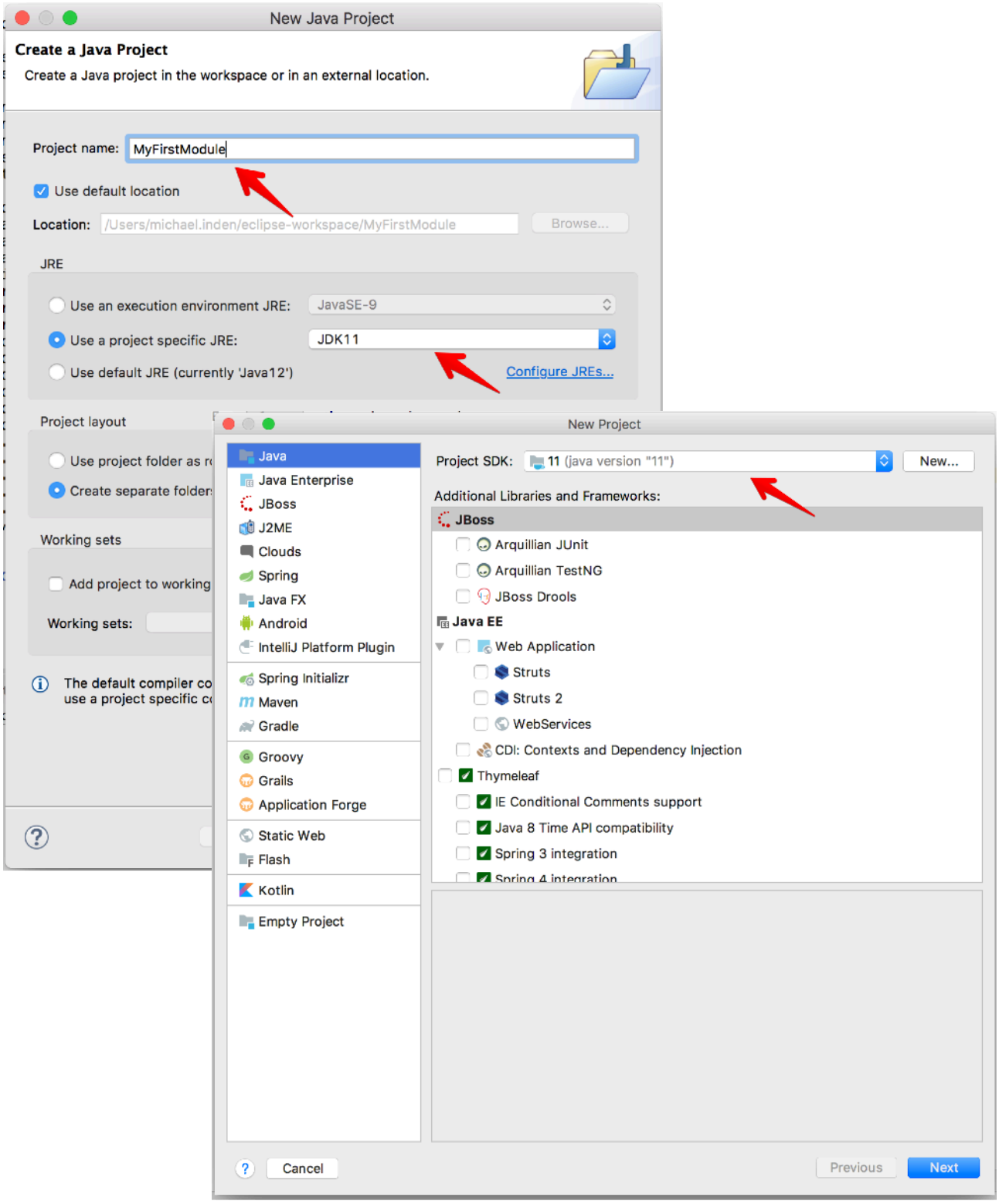
```
> tree executablemoduleexample/
executablemoduleexample/
├── bin
|   ├── jigsawapp
|   ├── java
|   └── keytool
├── conf
|   ├── net.properties
|   └── security
|       ├── java.policy
|       └── java.security
├── lib
|   ├── jli
|   |   └── libjli.dylib
|   ├── jspawnhelper
|   ├── jvm.cfg
...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

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

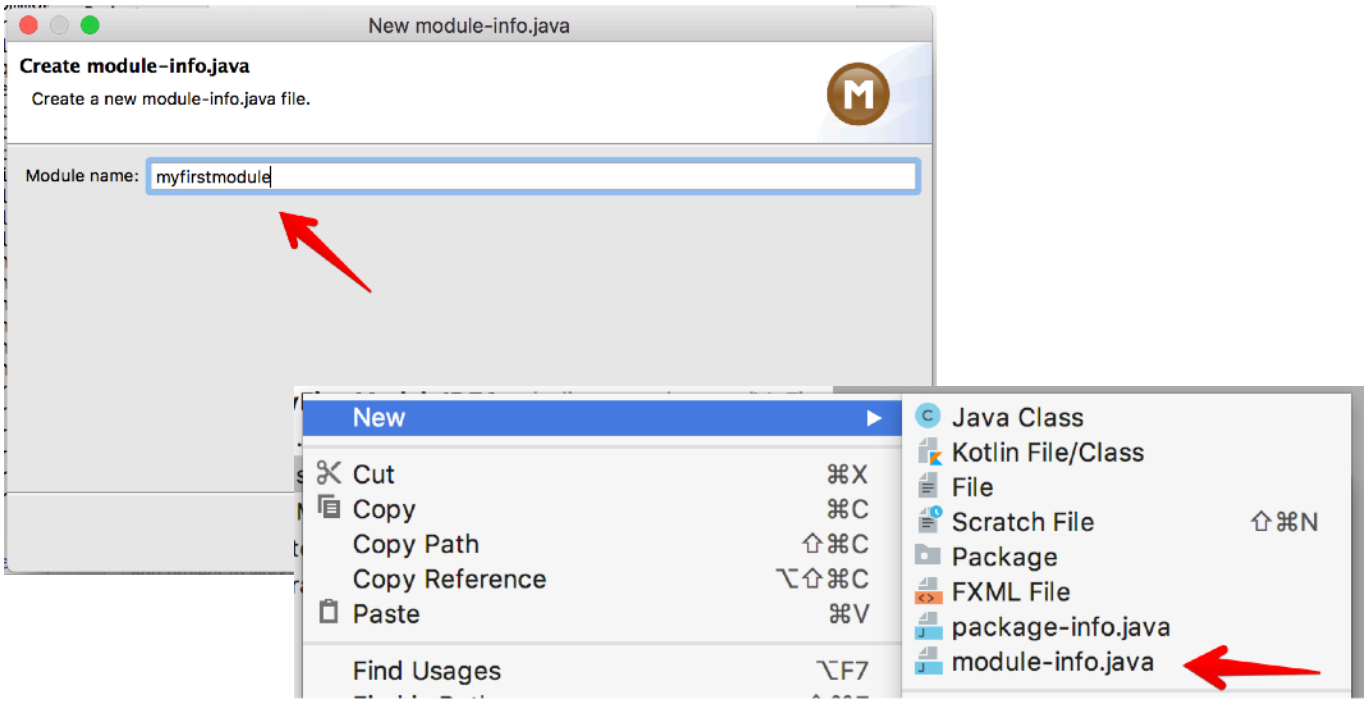
# IDE Version

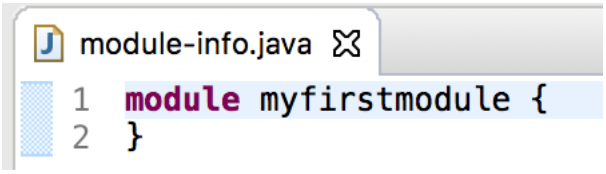
Use a current IDE of your choice. Below you can see Eclipse on the left and IntelliJ on the right.

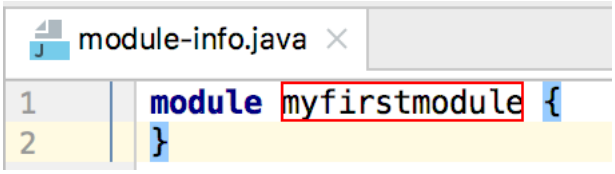
## 1) Creation of Java project (New > Java Project / New > Project)



## 2) Creation of Module Descriptor




## 3) Creation of directory and package hierarchy

New > Package com.hellojigsaw

## 4) Creation of application class

New > Class bzw. New > Java Class HelloJigsaw

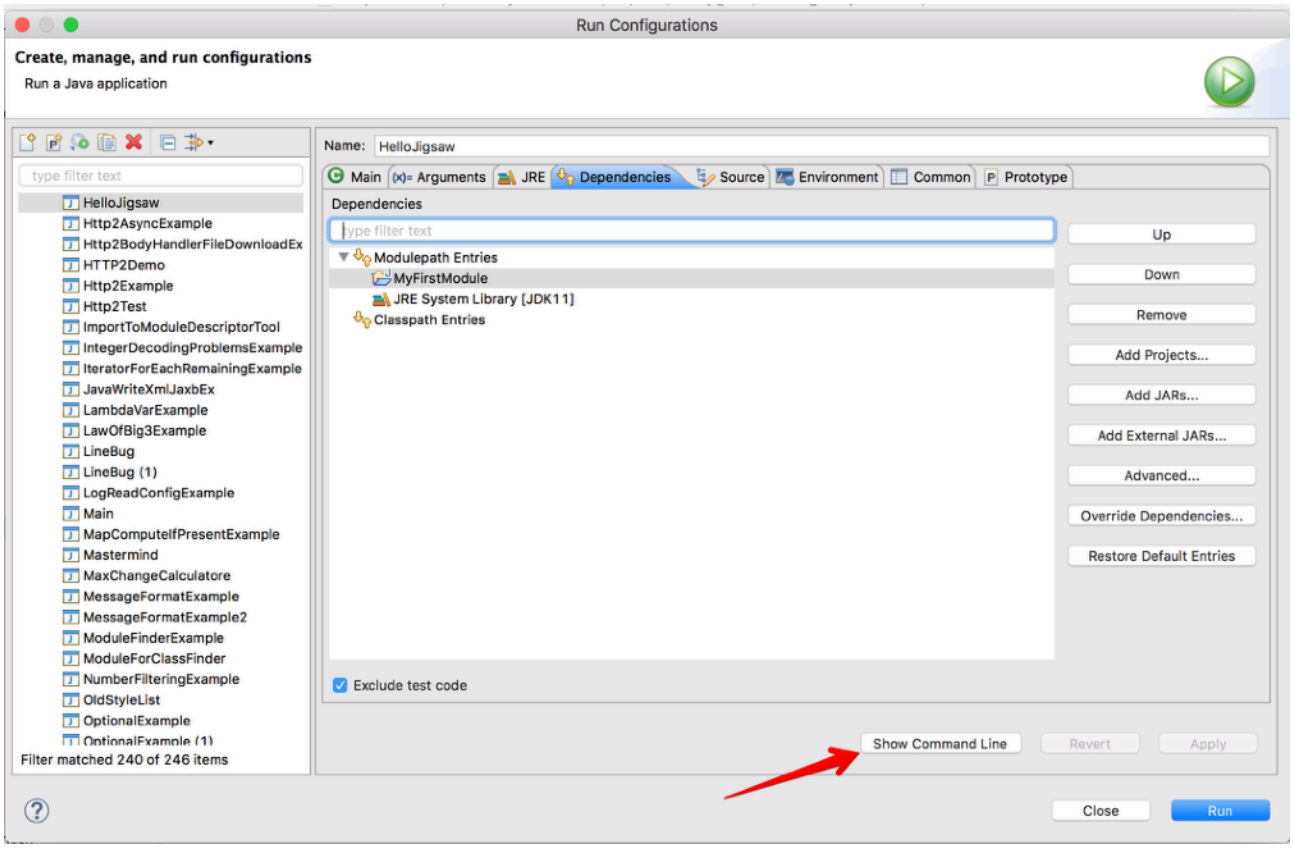
```
package com.hellojigsaw;

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

# 5) Start of application within a module

Start the program as usual from the IDE. Analyze then how to use -p ... -m ...

a) For IntelliJ this is easy because they are specified at startup.
b) For Eclipse there is the following trick:




Change to the project directory and try to start the application as follows:

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

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

To determine the output directories, use the `tree` command:

```
├── 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) Creation of deployable JAR

**Please note that with IDEs we do not use an "artificial" layer with module directories, as is unfortunately still suggested by Oracle ...**

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

**Parameters:**

```
--create => creation of archive (use –create here to avoid
misunderstandings with short -c / -C)
--file => archive file
-C => include files form the given directory
```

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

## 7) Check of dependencies

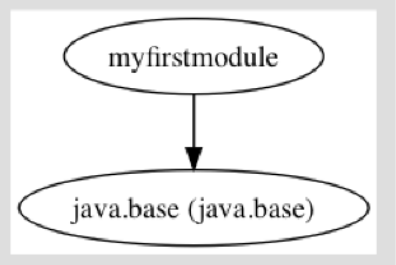
> jdeps lib/*.jar

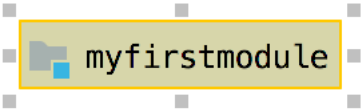
```
jdeps lib/*.jar
myfirstmodule
 [file:///Users/michael.inden/eclipse-workspace/MyFirstModule/lib/
myfirstmodule.jar]
   requires mandated java.base
myfirstmodule -> java.base
   com.hellojigsaw                                    -> java.io
java.base
   com.hellojigsaw                             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

## 8) Graphical representation

**ACTION: Installation of GRAPHVIZ, see console part**

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



# 9) Creation of Executable (Runtime Image)

**Check JDK version**

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

**Check Environment variable JAVA_HOME**

```
> echo $JAVA_HOME
```

**Setting it to version 11.0.2:**

```
>  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
```

**Alternatively:**

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

**Parameters:**

```
--module-path => path to modules
--add-modules => modules to include
--launcher => name of the class to execute
--output => output dir

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