Auslieferung eines Java-Programms

Anlage zu Informatik 2

Stand 22.06.2022

Erstellt von Tim Mühle

Inhalte

- jar-Archive
- Build-Tools
- Maven
- Installer erzeugen mit jpackage

Jar-Archive

- Jar-Dateien (abgeleitet von **J**ava-**Ar**chiv) sind zip-ähnliche Dateien mit der Dateiendung '*.jar*'
- Mit *Jar* können die einzelnen Programmteile als eine Datei zusammengepackt werden
- Anmerkung: „gepacktes Archiv“ ist nicht gleichbedeutend mit einer Komprimierung, sondern allgemein nur die Zusammenfassung von mehreren Dateien in einer gemeinsamen Datei
- Jar-Archive enthalten einen Ordner *META-INF* mit der Datei *MANIFEST.MF,* welche Metainformationen über das Archiv enthält. Unter anderem kann hier angegeben werden, wo sich die Main-Methode befindet, um die Jar-Datei ausführbar zu machen.
- Die meisten Betriebssysteme haben Jar-Dateien, also Dateien mit der Endung .jar, oft mit der JRE (Java Runtime Environment) verknüpft. Damit startet bei einem Doppelklick auf die Jar-Datei automatisch die enthaltene main-Methode (falls vorhanden und in MANIFEST.MF richtig eingestellt).
- Jar-Dateien können über die Konsole oder auch mit der Hilfe einer IDE, wie z.B. Eclipse, erzeugt werden

Build-Tools

- Besonders bei größeren Software-Projekten ist es nicht mehr praktikabel alle extern benötigten Jar-Dateien (Abhängigkeiten) manuell zu organisieren
- Auch das Erstellen eigener Jar-Dateien „per Hand“ ist bei entsprechender Häufigkeit ein aufwändiger Arbeitsschritt
- Build Tools sind Werkzeuge, die diese und andere Aufgaben automatisieren und erleichtern können
- Mit Build-Tools können z.B.
  - ➔ Abhängigkeiten verwaltet werden (bekannt von JavaFX)

    Vorteil: auch weitere benötigte Abhängigkeiten der eigenen Abhängigkeiten werden rekursiv aufgelöst (bindet man z.B.nur javafx.graphics ein, wird automatisch auch javafx.base heruntergeladen)
  - ➔ der Code automatisch kompiliert und zusammen mit weiteren Ressourcen als Jar gepackt werden

Maven

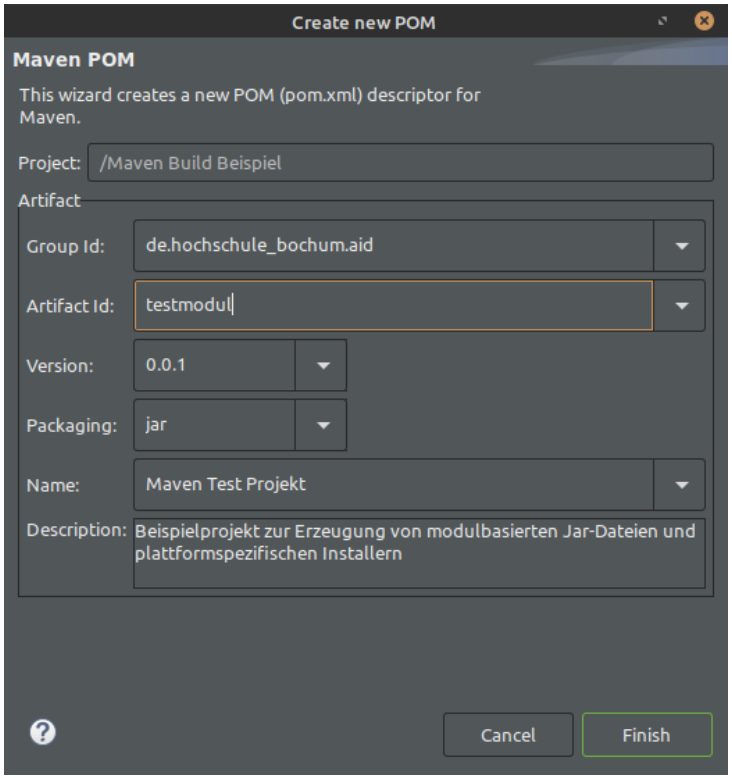
- Bekannte Vertreter für Build-Tools sind *Ant*, *Maven* und *Gradle*

  Im Weiteren soll hier näher auf das bereits bekannte Maven eingegangen werden. Die folgenden Schritte wären aber auch mit Gradle umsetzbar.

- Maven ist ein weit verbreitetes und kostenloses Build-Tool, welches von der Apache Software Foundation entwickelt wird und durch die freie Open-Source-Lizenz *Apache License 2.0* geschützt ist
- Bisher haben wir Maven genutzt, um die benötigten Abhängigkeiten („Dependencies“) für JavaFX in Eclipse zu integrieren
- Zudem kann Maven auch das Projekt durch Plugins als ausführbare Jar-Datei packen. Das nötige Vorgehen wird auf den folgenden Folien beschrieben.

Maven

Das folgende Beispielprojekt wird Ihnen als zip-Datei zur Verfügung gestellt.



1) Das Projekt muss wie bekannt in Eclipse zu einem Maven-Projekt umgewandelt werden

2) Für die „*Group ID*“ sollte nach Konvention die eigene Internet-Domain rückwärts angegeben werden, dies ist aber nicht zwingend erforderlich. Im Beispiel:

   de.hochschule_bochum.aid

   (Bindestriche sind problematisch, da diese bei Packagebezeichnungen nicht erlaubt sind)

3) Die „Artifact ID“ sollte dem Modulname des Projektes entsprechen

4) Die Version sollte mindestens 1.0.0 sein, um später auf allen Systemen Installer erzeugen zu können

5) Bei Interesse für die Vergabe von Versionsnummern sei folgende Seite empfohlen: https://semver.org/lang/de/

Maven

6) Anlegen der Projekstruktur:

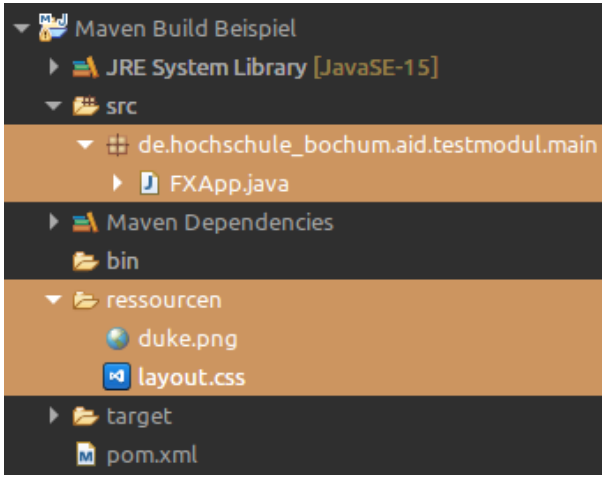
- Die Package-Hierarchie beginnt nach Konvention ebenfalls mit dem umgekehrten Domainnamen und das Hauptpaket sollte den Modulnamen haben. Im Beispiel ergibt sich also folgendes:

  de.hochschule_bochum.aid.testmodul

  (Bindestriche sind hier nicht erlaubt!)

- Die Hauptklasse mit der main-Methode soll im Beispiel *FXApp* heißen und im Paket *main* liegen

- Innerhalb des Projektes wird zusätzlich der Unterordner *ressourcen* für alle Dateien angelegt, die kein Programmcode sind. Beispielsweise können hier Bilder oder CSS-Dateien gespeichert werden.



Maven

7) Hinzufügen von Einstellungen in „*pom.xml*“:

   Zwischen der *description* und den *build*-Anweisungen werden einige Eigenschaften hinzugefügt:

```
...
<description>Beispiel Projekt zur Erzeugung von Jar Dateien und Installern</description>

<packaging>jar</packaging>

<properties>
    <javafx.version>16</javafx.version>
    <mainClass>de.hochschule_bochum.aid.testmodul.main.FXApp</mainClass>
    <dependenciesDir>${project.basedir}/installers/input/${version}/lib</dependenciesDir>
</properties>

<build>
    ...
```

- *javafx.version* ist die Variable für die gewünschte JavaFX (Haupt-)Version
- Bei *mainClass* muss bekanntgegeben werden, welche Klasse die main-Methode enthält, es müssen alle Pakete angegeben werden und die Endung *.java* entfällt
- *dependenciesDir* ist der Ordner, in den die Abhängigkeiten kopiert werden, damit diese später genutzt werden können

Maven

8) Anpassungen der Build-Anweisungen in „*pom.xml*“:

Um den Ordner *ressourcen* als zusätzliche Ressource hinzuzufügen, muss dieser innerhalb von *build* unterhalb von `<sourceDirectory>src</sourceDirectory>` ergänzt werden:

```
<build>
    <sourceDirectory>src</sourceDirectory>
    <resources>
        <resource>
            <directory>ressourcen</directory>
        </resource>
    </resources>
...
```

Die Angabe zwischen den *directory*-Tags ist dabei der Pfad zum gewünschten Ordner ausgehend vom Projekt-Ordner. Da der Ordner *ressourcen* bereits im Wurzelverzeichnis des Projektes liegt, genügt hier der Ordnername.

Durch diese Einstellung in der *pom.xml* werden die im Ordner *ressourcen* gespeicherten Dateien (z.B. eine CSS-Datei) mit in die erzeugte Jar-Datei gepackt. Somit können diese Dateien direkt über den Dateinamen ohne Pfadangabe in Java genutzt werden.

Es wäre auch möglich mehrere Ordner als Ressource hinzuzufügen, z.B. um Bilder und css-Dateien zu trennen. Dann müssen entsprechend mehrere *resource*-Tags mit der Angabe der *directory* innerhalb des Blocks *resources* hinzugefügt werden.

Maven

Damit Maven eine ausführbare Jar-Datei erzeugt, die alle Ressourcen beinhaltet, muss zwischen `<plugins>` und `</plugins>` ein weiteres Plugin (unter dem Compiler-Plugin) ergänzt werden:

```
<plugins>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    ...
  </plugin>

  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>3.2.0</version>
    <configuration>
      <outputDirectory>${project.basedir}/installers/input/${version}</outputDirectory>
      <archive>
        <manifest>
          <mainClass>${mainClass}</mainClass>
        </manifest>
      </archive>
    </configuration>
  </plugin>

</plugins>
```

Maven

Die Dependencies können mit einem weiteren Plugin in einen Ordner neben der Jar-Datei kopiert werden:

```
      ...
 </plugin>

 <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-dependency-plugin</artifactId>
   <version>2.10</version>
   <configuration>
     <outputDirectory>${dependenciesDir}</outputDirectory>
   </configuration>
   <executions>
     <execution>
       <id>copy-dependencies</id>
       <phase>package</phase>
       <goals>
         <goal>copy-dependencies</goal>
       </goals>
       <configuration>
         <outputDirectory>${dependenciesDir}</outputDirectory>
         <overWriteReleases>true</overWriteReleases>
         <overWriteSnapshots>true</overWriteSnapshots>
         <overWriteIfNewer>true</overWriteIfNewer>
       </configuration>
     </execution>
   </executions>
 </plugin>
</plugins>
```

Maven

9) Zum Schluss werden wieder die Abhängigkeiten für JavaFX zwischen den schließenden Tags `</build>` und `</project>` ergänzt:

```
    ...
</build>

<dependencies>
    <!-- javaFX -->
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-base</artifactId>
        <version>${javafx.version}</version>
    </dependency>

    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>${javafx.version}</version>
    </dependency>

    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-graphics</artifactId>
        <version>${javafx.version}</version>
    </dependency>
</dependencies>

</project>
```

Durch die Nutzung der Variable `${javafx.version}` kann die Version für JavaFX schnell in den Properties angepasst werden.

Maven

- Das Testprogramm in „FXApp.java“:

```
package de.hochschule_bochum.aid.testmodul.main;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class FXApp extends Application {

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) throws Exception {
        StackPane root = new StackPane();
        root.getChildren().add(new ImageView(new Image("duke.png")));

        Scene sc = new Scene(root, 800, 800);
        sc.getStylesheets().add("layout.css");

        primaryStage.setScene(sc);
        primaryStage.setTitle("Maven Beispiel");
        primaryStage.show();
    }
}
```

Maven

- Die Modulbeschreibung in „module-info.java“:

```
module testmodul {
    requires transitive javafx.graphics;

    exports de.hochschule_bochum.aid.testmodul.main;
}
```

- Layout.css:

```
.root {
    -fx-background-color: lightskyblue;
}
```

Maven

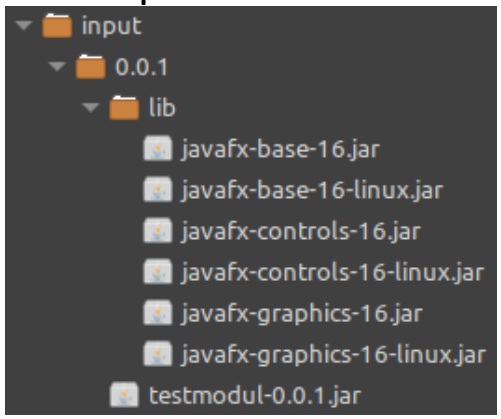
- Zudem muss Maven installiert werden, damit das Tool auf der Kommandozeile ausgeführt werden kann:

  http://maven.apache.org/install.html

- Die Installation von Maven ist ähnlich zu der Installation des openJDK: Wie schon bei dem openJDK muss auch Maven zu der PATH-Variablen des Systems hinzugefügt werden
- Jetzt kann im Projektordner, in dem sich auch pom.xml befindet, eine neue Kommandozeile geöffnet werden
- Durch den Befehl "**`mvn clean install`**" wird das Projekt kompiliert und als Jar-Archiv gepackt
- Im Projektordner ist nun der Ordner *installers* hinzugekommen. In dem Verzeichnis

  installers > input > VERSIONSNUMMER

  befinden sich die generierte Jar-Datei und der Ordner *lib* mit den plattformabhängigen JavaFX-Dateien.



Maven

- Öffnet man nun die Konsole in dem Ordner mit der jar-Datei (im Beispiel der Ordner 0.0.1), kann das Programm mit dem Befehl

```
java --module-path lib --add-modules ALL-MODULE-PATH -jar testmodul-0.0.1.jar
```

  ausgeführt werden. (Die Angabe `0.0.1` ist dabei abhängig von der Version in *pom.xml*)
- Da der Befehl sehr unhandlich ist, kann dieser auch in einem ausführbaren Startskript gespeichert werden. Dies ist allerdings für jedes Betriebssystem einzeln nötig
- Zudem sind die Jar-Archive von JavaFX plattformabhängig, daher muss der sogenannte Build-Prozess mit Maven für jedes Betriebssystem separat erfolgen
- Wenn das Programm in dieser Form z.B. als zip-Datei mit Startskript weitergegeben werden soll, muss also für jedes Betriebssystem eine eigene zip-Datei mit den entsprechenden Abhängigkeiten erzeugt und bereitgestellt werden

Installer erzeugen mit jPackage

- Für die meisten Nutzer:innen ist Java und vor allem die Konsole etwas unbekanntes. Besser wäre, wenn für jedes Betriebssystem statt der Jar-Datei und dem Ordner lib, die bisher nur umständlich ausgeführt werden können, ein Installer weitergegeben werden kann.
- Im JDK gibt das Kommandozeilentool jpackage, welches mit Java 14 testweise und mit Java 16 final eingeführt wurde
- Mit jpackage steht nun ein mächtiges Tool zur Verfügung, um ein Java-Programm leicht durch Installer für Windows, macOS und Linux auszuliefern
- Die Installer enthalten nicht nur das Programm, sondern auch die benötigte JRE (Java-Runtime-Environment). Auf dem Zielsystem muss somit kein Java installiert sein. Die JRE wird dabei auf die nötigsten Module reduziert, um Ressourcen zu sparen.
- Somit beinhaltet jedes Programm immer eine kompatible JRE, um unabhängig von installierten Java Versionen lauffähig zu sein
- Besonders bei älteren JavaFX Programmen ist dies hilfreich, da JavaFX zum Zeitpunkt von Java 8 noch zur JavaSE (**S**tandard **E**dition) gehörte (also nicht eigenständig war) und diese daher nicht mit neueren Java Versionen ausgeführt werden können
- Ein Nachteil ist, dass beim Einsatz vieler solcher Programme die benötigte Basis der JRE mehrfach vorhanden ist und mehr Speicherplatz belegt wird
- Wichtig: Installer können immer nur auf der Zielplattform erzeugt werden, da plattform-spezifische Tools benötigt werden

Installer erzeugen mit jPackage - Überblick

- jpackage kann mit vielen Optionen aufgerufen werden, zu den generellen Optionen gehören:

Option	Beschreibung
`--type <typ>`	Was soll erzeugt werden? ("app-image", "exe", "msi", "rpm", "deb", "pkg", "dmg" → "dmg" benötigt eine Signierung!)
`--name <name>`	Name des Programms
`--description <descr. str.>`	Beschreibung, die im Installer angezeigt wird
`--app-version <version>`	Programmversion (sollte der Version in pom.xml entsprechen)
`--vendor <vendor string>`	Hersteller des Programms
`--copyright <copyr. Str.>`	Hinweis auf ein Copyright
`--license-file <file>`	Datei mit dem Lizenztext
`--input <input path>`	Eingabeordner der alle Programmteile enthält
`--dest <output path>`	Ordner, in dem die Installer gespeichert werden
`--icon <icon file path>`	Programm-Icon (Dateityp muss .ico für Windows, .icns für macOS, .png für Linux sein)

- Die Platzhalter wie *<name>* sind durch spezifische Werte zu ersetzen

Installer erzeugen mit jPackage - Überblick

- Für modulbasierte Projekte sind folgende Angaben wichtig:

Option	Beschreibung
`--module-path <module path>`	Pfad zu einem oder mehreren Modulen, die beim Start des Programms dem Modulpfad hinzugefügt werden
`--add-modules <module name>`	Angabe der Module, die beim Start zusätzlich aus dem Modulpfad geladen werden
`--module <module name>`	Angabe des Hauptmoduls mit der Main-Methode (optional kann nach dem Modulnamen ein / und dann der Pfad zu der Hauptklasse angegeben werden)

Installer erzeugen mit jPackage - Windows

- Auf Windows-Systemen sind folgende Optionen interessant:

Option	Beschreibung
`--win-dir-chooser`	Der Nutzer kann den Installationspfad einstellen
`--win-menu`	Das Programm wird im Startmenü angezeigt
`--win-menu-group <menu group name>`	Startmenü-Gruppe (Unterordner im Startmenü), in der das Programm durch --win-menu hinzugefügt wird
`--win-per-user-install`	Die Installation erfolgt nur für einen Nutzer und nicht systemweit
`--win-shortcut`	Es wird ein Link auf dem Desktop angelegt

Installer erzeugen mit jPackage - macOS

- Bei macOS stehen diese zusätzlichen Optionen zur Auswahl:

Option	Beschreibung
`--mac-package-name <name string>`	Name, der in der macOS Menübar angezeigt wird (darf maximal 16 Zeichen lang sein, standardmäßig der Wert von `--name`)
`--mac-package-identifier <string>`	Eindeutige Identifizierung für das Paket. Standardmäßig gleich mit dem Klassennamen inklusive Package-Pfad, darf aber keine Unterstriche enthalten!

- Darüber hinaus gibt es weitere Optionen, um den Installer zu „signieren“, also um zu markieren, dass das Paket von einer vertrauenswürdigen Quelle stammt. Dies ist insbesondere bei dem Pakettyp *dmg* nötig, da sonst die Installation fehlschlägt. Allerdings funktioniert das Signieren nur, wenn man offiziell als Entwickler bei Apple eingetragen ist, was mit einer jährlichen Gebühr verbunden ist.

Installer erzeugen mit jPackage - Linux

- Für Linux sind folgende Optionen verfügbar:

Option	Beschreibung
`--linux-menu-group <group-name>`	Menügruppe, in der sich diese Anwendung befindet.
`--linux-app-category <category>`	Gruppenwert der Datei RPM <Name>.spec oder Sektionswert der DEB-Steuerdatei.
`--linux-shortcut`	Erzeugt eine Verknüpfung für die Anwendung
`--linux-rpm-license-type`	Name der Lizenz, unter der das Programm veröffentlicht wird

Installer erzeugen mit jPackage - Beispiel

- Weitere optionale Parameter sind ausführlich auf der offiziellen Seite dokumentiert:

  https://docs.oracle.com/en/java/javase/16/docs/specs/man/jpackage.html
- In der Praxis ist es nicht sinnvoll den Befehl per Hand mit allen Optionen in die Konsole einzugeben
- Praktikabler ist es, ein ausführbares Skript für die Konsole zu erstellen (.bat auf Windows, .command auf macOS, .sh auf Linux)
- Vorlagen für dieses Beispielprojekt finden Sie auf den nächsten Seiten und auch im bereitgestellten Projektordner
- In den Vorlagen sind zu Beginn Variablen deklariert, die von jpackage genutzt werden. So kann das Skript leicht an neue Projekte angepasst werden.
- **Die Versionsnummer im Skript muss immer dem Stand in *pom.xml* entsprechen und daher bei Änderung in allen Skripten und in pom.xml angepasst werden.**

Installer erzeugen mit jPackage – Skript Windows

Batch-Skript für Windows

```
@echo off
rem ---- Benoetigte Informationen in Variablen speichern -------------
rem Mit den folgeneden Variablen koennen die Grundlegenden Daten fuer
rem das Projekt eingestellt werden:
set NAME="Maven Build Beispiel"
set DESCRIPTION="Beispielprojekt zur Erzeugung von modulbasierten Jar-
Dateien und plattformspezifischen Installern"
set VERSION=1.0.0
set VENDOR="NAME DES ENTWICKLERS EINTRAGEN"
set COPYRIGHT="Copyright (c)"
set LICENSE_FILE="LICENSE.txt"


rem ---- Einstellungen fuer jpackage: --------------------------------
set INPUT="installers/input/%VERSION%"
set OUT="installers/%VERSION%/Windows"
set MODULE_PATH="installers/input/%VERSION%/lib"
set MAIN_MODULE="testmodul"
set MAIN_CLASS="de.hochschule_bochum.aid.testmodul.main.FXApp"


rem Systemspezifische Optionen
set WIN_MENU_GROUP="Maven Build Beispiel"


rem ---- Eingabeordner fuer jpackage leeren --------------------------
if exist %INPUT% del /Q %INPUT%


rem ---- Maven build -------------------------------------------------
echo.
echo Maven build durchfuehren
echo.
call mvn clean install

rem ---- Installer erzeugen ------------------------------------------
echo.
echo Installer fuer Windows werden erzeugt.
echo.


rem ---- exe ----
echo %NAME%-%VERSION%-win-install.exe wird erstellt
echo.
```

```
@echo on
jpackage ^
--type exe ^
--name %NAME% ^
--description %DESCRIPTION% ^
--app-version %VERSION% ^
--vendor %VENDOR% ^
--copyright %COPYRIGHT% ^
--license-file %LICENSE_FILE% ^
--input %INPUT% ^
--dest %OUT% ^
--module-path %MODULE_PATH% ^
--module-path "%INPUT%" ^
--module "%MAIN_MODULE%/%MAIN_CLASS%" ^
--win-dir-chooser ^
--win-shortcut ^
--win-menu ^
--win-menu-group %WIN_MENU_GROUP%
@echo off

ren %OUT%\%NAME%-%VERSION%.exe %NAME%-%VERSION%-win-install.exe
echo.
echo.

rem ---- msi ----
echo %NAME%-%VERSION%-win-install.msi wird erstellt
echo.
@echo on

jpackage ^
--type msi ^
--name %NAME% ^
--description %DESCRIPTION% ^
--app-version %VERSION% ^
--vendor %VENDOR% ^
--copyright %COPYRIGHT% ^
--license-file %LICENSE_FILE% ^
--input %INPUT% ^
--dest %OUT% ^
--module-path %MODULE_PATH% ^
--module-path "%INPUT%" ^
--module "%MAIN_MODULE%/%MAIN_CLASS%" ^
--win-dir-chooser ^
--win-shortcut ^
--win-menu ^
--win-menu-group %WIN_MENU_GROUP%


@echo off
ren %OUT%\%NAME%-%VERSION%.msi %NAME%-%VERSION%-win-install.msi
echo.
echo.
rem ---- Auf Bestaetigung von Benutzer warten ------------------------------------
echo "Zum Abschliessen eine beliebige Taste druecken"
pause
```

Installer erzeugen mit jPackage – Skript macOS

Skript für macOS

```
#!/bin/sh
cd "$(dirname "$0")" # zum Pfad dieses Skriptes wechseln


# ---- Benoetigte Informationen in Variablen speichern --------------------------


# Mit den folgeneden Variablen koennen die Grundlegenden Daten fuer das Projekt
# eingestellt werden:
NAME="Maven Build Beispiel"
DESCRIPTION="Beispielprojekt zur Erzeugung von modulbasierten Jar-Dateien und
plattformspezifischen Installern"
VERSION="1.0.0"     # Version muss groesser gleich 1.0.0 sein!
VENDOR="NAME DES ENTWICKLERS EINTRAGEN"
COPYRIGHT="Copyright ©"
LICENSE_FILE="LICENSE.txt"


# Einstellungen fuer jpackage:
INPUT="installers/input/${VERSION}"
OUT="installers/${VERSION}/macOS"
MODULE_PATH="installers/input/${VERSION}/lib"
MAIN_MODULE="testmodul"
MAIN_CLASS="de.hochschule_bochum.aid.testmodul.main.FXApp"


# Weitere Befehle fuer jpackage:
# App Icon aendern: --icon "path/to/icon.png"
#ICON="path/to/icon.png"


# Systemspezifische Optionen
# Paketname darf maximal 16 Zeichen haben!
MAC_PACKAGE_NAME="Build Beispiel"


# weltweit eindeutiger Paketidentifier -> Pfad zur Main-Klasse ohne Unterstrich!
MAC_PACKAGE_ID="de.hochschule-bochum.aid.testmodul.main.FXApp"


# ---- Eingabeordner leeren -----------------------------------------------------
mkdir -p $INPUT
rm -r -f $INPUT/*


# ---- Maven build --------------------------------------------------------------
echo ""
echo "Maven build durchfuehren"
echo ""
mvn clean install

# ---- Installer erzeugen -------------------------------------------------------
echo ""
echo "Installer fuer macOS werden erzeugt."
echo ""
```

```
echo "App-Image wird erstellt"
echo ""
jpackage \
--type "app-image" \
--name "${NAME}" \
--description "${DESCRIPTION}" \
--app-version "${VERSION}" \
--vendor "${VENDOR}" \
--copyright "${COPYRIGHT}" \
--input "${INPUT}" \
--dest "${OUT}" \
--module-path "${MODULE_PATH}" \
--module-path "${INPUT}" \
--module "${MAIN_MODULE}/${MAIN_CLASS}" \
--mac-package-name "${MAC_PACKAGE_NAME}" \
--mac-package-identifier "${MAC_PACKAGE_ID}" \
#--icon "${ICON}"

echo ""
echo "${NAME-$VERSION}.pkg wird erstellt"
echo ""
jpackage \
--type "pkg" \
--name "${NAME}" \
--description "${DESCRIPTION}" \
--app-version "${VERSION}" \
--vendor "${VENDOR}" \
--copyright "${COPYRIGHT}" \
--license-file "${LICENSE_FILE}" \
--input "${INPUT}" \
--dest "${OUT}" \
--module-path "${MODULE_PATH}" \
--module-path "${INPUT}" \
--module "${MAIN_MODULE}/${MAIN_CLASS}" \
--mac-package-name "${MAC_PACKAGE_NAME}" \
--mac-package-identifier "${MAC_PACKAGE_ID}" \
#--icon "${ICON}"

echo ""
echo ""
echo "Der Prozess wurde beendet." echo ""
```

Achtung: Die minimale erlaubte Version für macOS ist 1.0.0. Ist die Hauptversion kleiner als 1, wird jPackage mit einem Fehler beendet. Die Version in pom.xml und den verschiedenen Skripten müssen immer übereinstimmen.

Installer erzeugen mit jPackage – Skript DEB-Linux

Skript für ein Debian-basiertes Linux (z.B. Mint)

```
#!/bin/sh
cd "$(dirname "$0")" # zum Pfad dieses Skriptes wechseln
# ---- Benoetigte Informationen in Variablen speichern --------------------------

# Mit den folgeneden Variablen koennen die Grundlegenden Daten fuer das Projekt
# eingestellt werden:
NAME="Maven Build Beispiel"
DESCRIPTION="Beispielprojekt zur Erzeugung von modulbasierten Jar-Dateien und
plattformspezifischen Installern"
VERSION=1.0.0
VENDOR="NAME DES ENTWICKLERS EINTRAGEN"
COPYRIGHT="Copyright ©"
LICENSE_FILE="LICENSE.txt"

# Einstellungen fuer jpackage:
INPUT="installers/input/${VERSION}"
OUT="installers/${VERSION}/Linux"
MODULE_PATH="installers/input/${VERSION}/lib"
MAIN_MODULE="testmodul"
MAIN_CLASS="de.hochschule_bochum.aid.testmodul.main.FXApp"

# Weitere Befehle fuer jpackage:
# App Icon aendern: --icon "path/to/icon.png"
#ICON="path/to/icon.png"

# Systemspezifische Optionen
LINUX_MENU_GROUP="Development;Education;"
LINUX_RPM_LICENSE_TYPE="MIT"

# ---- Eingabeordner leeren -----------------------------------------------------
mkdir -p $INPUT
rm -r -f $INPUT/*

# ---- Maven build --------------------------------------------------------------
echo ""
echo "Maven build durchfuehren" echo ""
mvn clean install
```

```
# ---- Installer erzeugen -------------------------------------------------------
echo ""
echo "Installer fuer Linux werden erzeugt."
echo ""

echo "App-Image wird erstellt"
echo ""
jpackage \
--type "app-image" \
--name "${NAME}" \
--description "${DESCRIPTION}" \
--app-version "${VERSION}" \
--vendor "${VENDOR}" \
--copyright "${COPYRIGHT}" \
--input "${INPUT}" \
--dest "${OUT}" \
--module-path "${MODULE_PATH}" \
--module-path "${INPUT}" \
--module "${MAIN_MODULE}/${MAIN_CLASS}" \
# --icon "${ICON}"

echo "${NAME-$VERSION}.deb wird erstellt"
echo ""
jpackage \
--type "deb" \
--name "${NAME}" \
--description "${DESCRIPTION}" \
--app-version "${VERSION}" \
--vendor "${VENDOR}" \
--copyright "${COPYRIGHT}" \
--license-file "${LICENSE_FILE}" \
--input "${INPUT}" \
--dest "${OUT}" \
--module-path "${MODULE_PATH}" \
--module-path "${INPUT}" \
--module "${MAIN_MODULE}/${MAIN_CLASS}" \
--linux-menu-group "${LINUX_MENU_GROUP}" \
--linux-shortcut \
--linux-rpm-license-type "${LINUX_RPM_LICENSE_TYPE}" \
# --icon "${ICON}"

echo ""
echo "Der Prozess wurde beendet."
echo ""
```

Installer erzeugen mit jPackage – Skript RPM-Linux

Skript für ein auf RedHat basierendes Linux (z.B. CentOS oder Fedora)

```
#!/bin/sh
cd "$(dirname "$0")" # zum Pfad dieses Skriptes wechseln
# ---- Benoetigte Informationen in Variablen speichern --------------------------

# Mit den folgeneden Variablen koennen die Grundlegenden Daten fuer das Projekt
# eingestellt werden:
NAME="Maven Build Beispiel"
DESCRIPTION="Beispielprojekt zur Erzeugung von modulbasierten Jar-Dateien und
plattformspezifischen Installern"
VERSION=1.0.0
VENDOR="NAME DES ENTWICKLERS EINTRAGEN"
COPYRIGHT="Copyright ©"
LICENSE_FILE="LICENSE.txt"

# Einstellungen fuer jpackage:
INPUT="installers/input/${VERSION}"
OUT="installers/${VERSION}/Linux"
MODULE_PATH="installers/input/${VERSION}/lib"
MAIN_MODULE="testmodul"
MAIN_CLASS="de.hochschule_bochum.aid.testmodul.main.FXApp"

# Weitere Befehle fuer jpackage:
# App Icon aendern: --icon "path/to/icon.png"
#ICON="path/to/icon.png"

# Systemspezifische Optionen
LINUX_MENU_GROUP="Development;Education;"
LINUX_RPM_LICENSE_TYPE="MIT"

# ---- Eingabeordner leeren -----------------------------------------------------
mkdir -p $INPUT
rm -r -f $INPUT/*

# ---- Maven build --------------------------------------------------------------
echo ""
echo "Maven build durchfuehren" echo ""
mvn clean install
```

```
# ---- Installer erzeugen -------------------------------------------------------
echo ""
echo "Installer fuer Linux werden erzeugt."
echo ""

echo "App-Image wird erstellt"
echo ""
jpackage \
--type "app-image" \
--name "${NAME}" \
--description "${DESCRIPTION}" \
--app-version "${VERSION}" \
--vendor "${VENDOR}" \
--copyright "${COPYRIGHT}" \
--input "${INPUT}" \
--dest "${OUT}" \
--module-path "${MODULE_PATH}" \
--module-path "${INPUT}" \
--module "${MAIN_MODULE}/${MAIN_CLASS}" \
# --icon "${ICON}"

echo "${NAME-$VERSION}.rpm wird erstellt"
echo ""
jpackage \
--type "rpm" \
--name "${NAME}" \
--description "${DESCRIPTION}" \
--app-version "${VERSION}" \
--vendor "${VENDOR}" \
--copyright "${COPYRIGHT}" \
--license-file "${LICENSE_FILE}" \
--input "${INPUT}" \
--dest "${OUT}" \
--module-path "${MODULE_PATH}" \
--module-path "${INPUT}" \
--module "${MAIN_MODULE}/${MAIN_CLASS}" \
--linux-menu-group "${LINUX_MENU_GROUP}" \
--linux-shortcut \
--linux-rpm-license-type "${LINUX_RPM_LICENSE_TYPE}" \
# --icon "${ICON}"

echo ""
echo "Der Prozess wurde beendet."
echo ""
```

Installer erzeugen mit jPackage

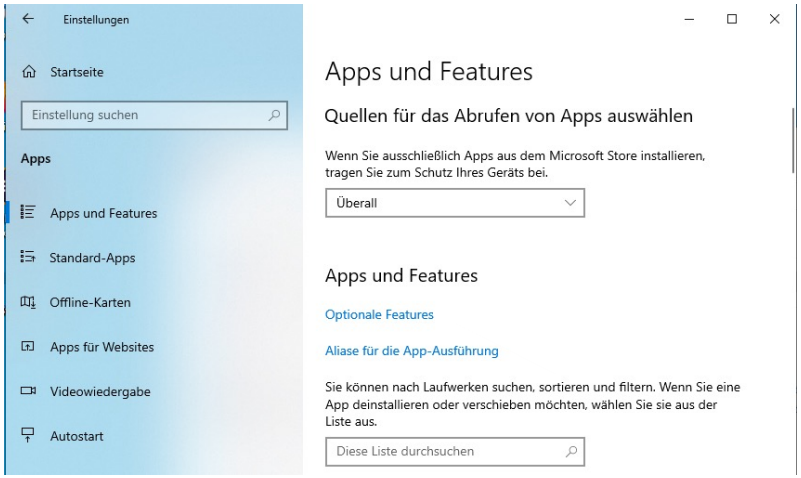
- Diese Skripte müssen in dem Wurzelverzeichnis des Projektes abgelegt werden, in dem sich auch *pom.xml* befindet, damit diese korrekt funktionieren
- Zudem muss in dem Wurzelverzeichnis auch die Datei "LICENSE.txt" angelegt werden, die den Lizenztext enthält, der dem Installer hinzugefügt wird
- Dieser Lizenz muss der/die Nutzer:in bei der Installation zustimmen
- Im Beispiel wurde die freie Open-Source Lizenz „MIT-License“ gewählt. Eine kompakte Übersicht der gängigsten Open-Source Lizenzen gibt es hier:

  https://choosealicense.com/licenses/

  Es ist aber auch jede andere Lizenz möglich.
- **Achtung bei Wahl einer anderen Lizenz:** In den Skripten für Linux muss die Variable *LINUX_RPM_LICENSE_TYPE* an die genutzte Lizenz angepasst werden

Installer erzeugen mit jPackage

- Je nach System sind ggf. weitere Programme nötig, um die Installer zu erzeugen
  - Windows: Das WiX-Toolset (https://wixtoolset.org) muss installiert sein
    - Um dieses installieren zu können muss in Windows das .NET Framework 3.5 aktiviert werden
    - Dazu müssen über das Startmenü die Windows-Einstellungen geöffnet und dann der Punkt *Apps* gewählt werden
    - Hier muss unter der Überschrift „Apps und Features“ der Link „*Optionale Features*“ geöffnet werden

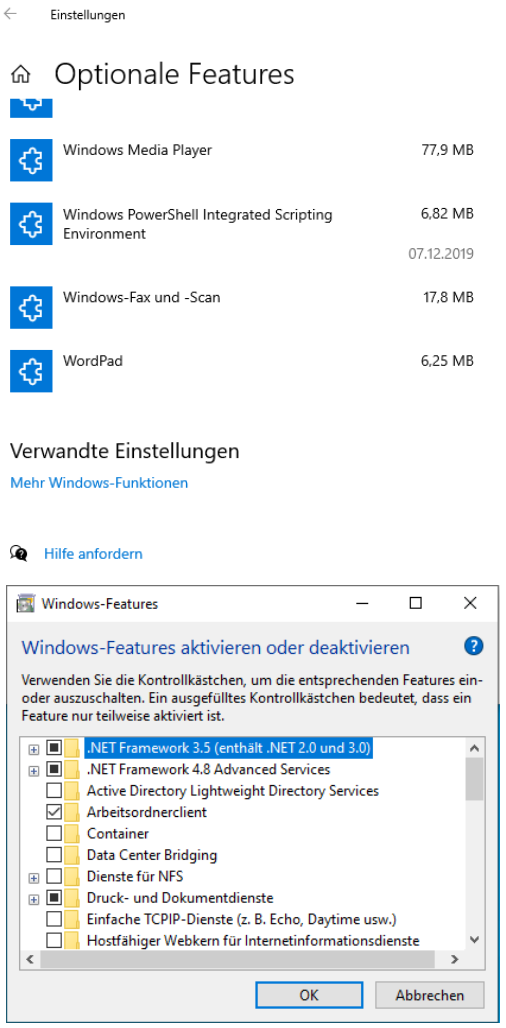


Installer erzeugen mit jPackage

- Scrollt man nun nach ganz unten kann der Link „*Mehr Windows-Funktionen*“ geöffnet werden
- In dem neuen Fenster muss nun ein Haken bei „*.NET Framework 3.5*“ gesetzt werden
- Das Fenster kann mit *OK* geschlossen werden
- Die erforderlichen Komponenten werden nun heruntergeladen, dies muss noch einmal bestätigt werden
- Danach kann das WiX-Toolset installiert werden

- Welche Pakete für Linux erforderlich sind, ist von der gewählten Distribution abhängig
  - Mint benötigt keine weiteren Installationen zum Erzeugen von deb-Paketen
  - Um mit CentOS 7 einen rpm-Installer zu erzeugen, muss zuvor folgender Befehl einmlig im Terminal ausgeführt werden:

```
sudo yum install rpm-build
```



Installer erzeugen mit jPackage

- Für das Erstellen von pkg-Installern mit macOS ist keine weitere Software notwendig, es müssen aber folgende Punkte beachtet werden:
  - Die Version (Option `--app-version`) muss mindestens 1.0.0 sein. Ist die Hauptversion kleiner als 1, wird jPackage mit einem Fehler beendet
  - Der Package-Identifier (Option `--mac-package-identifier`) darf im Gegensatz zur Packagebezeichnung in Java keine Unterstriche, stattdessen aber Bindestriche enthalten
  - Der Package-Name (Option `--mac-package-name`) darf maximal 16 Zeichen lang sein. Diese Beschränkung wurde von Apple eingeführt, da der Name in der nativen macOS-Menübar angezeigt wird und dort nicht zu viel Platz einnehmen soll.
- Die macOS dmg-Installer sind ohne Signierung nicht funktionsfähig, würden aber weitere Entwickler-Tools von Apple benötigen. Diese können entweder beim ersten Versuch einen dmg-Installer zu erzeugen oder durch die Installation von Xcode (Apples IDE) installiert werden.

Installer erzeugen mit jPackage

- Wie bereits erwähnt muss jedes Skript auf der jeweiligen Zielplattform ausgeführt werden
- Eine mögliche Lösung ist es, die Installer innerhalb einer virtuellen Maschine (z.B. über VirtualBox oder VMWare) zu erzeugen, damit nicht jedes Mal der PC oder das Betriebssystem gewechselt werden muss
- Auf Linux muss das Skript nach dem Erstellen zunächst mit dem Befehl

  `chmod +x dateiname.sh`

  ausführbar gemacht werden
- Unter macOS ist dazu

  `chmod +x dateiname.command`

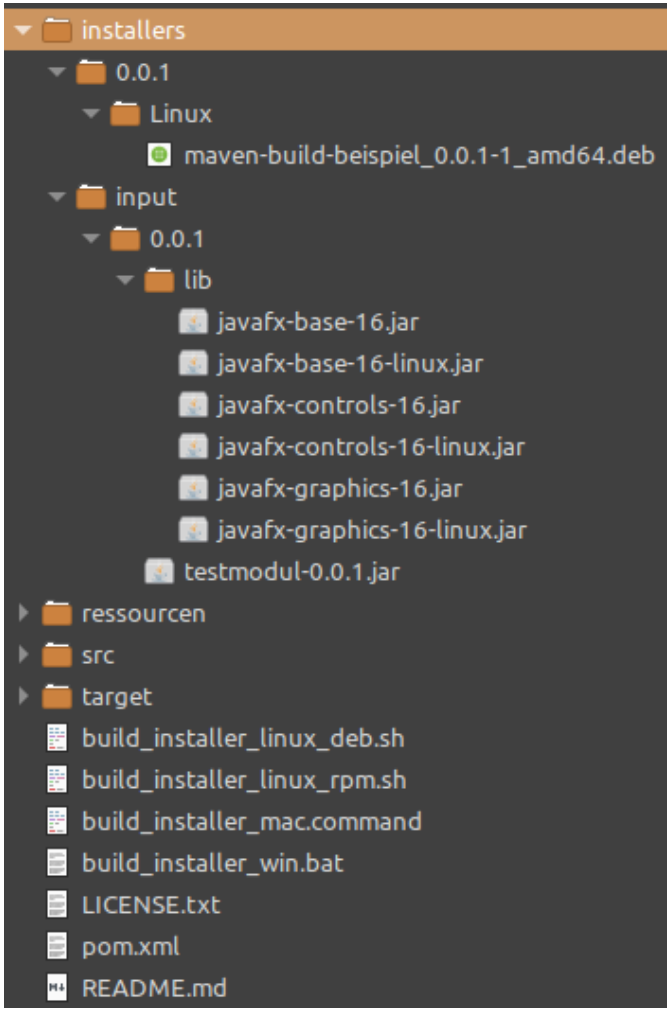
  nötig
- Die Skripte können entweder mit einem Doppelklick oder über die Konsole ausgeführt werden. Befindet man sich mit der Konsole im Stammverzeichnis des Projektes (in dem sich die Installer-Skripte und pom.xml befinden), lauten die Befehle zum Ausführen der Skripte wie folgt:
  - Mit der cmd oder PowerShell in Windows: "`.\dateiname.bat`"
  - Mit einem Linux Terminal: "`./dateiname.sh`"
  - Mit einem macOS Terminal: "`./dateiname.command`"

  Statt *dateiname* muss der korrekte Name des Skriptes eingegeben werden!

Installer erzeugen mit jPackage – Das Ergebnis

Beim Ausführen der Skripte werden im Stammordner des Projektes verschiedene Ordner erzeugt:

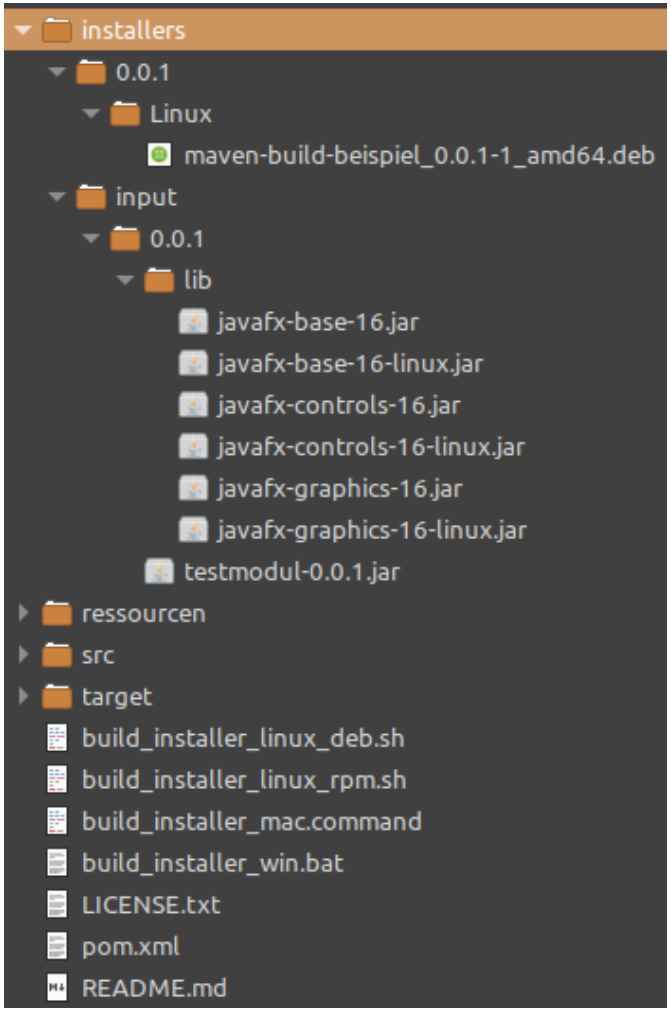
- Maven legt den Ordner *target* an, dieser ist nur für Maven und Eclipse wichtig.
- Der Unterordner *input* in *installers* enthält einen zur Version passenden Ordner mit der von Maven erzeugten jar-Datei und dem Ordner *lib* mit Kopien der Dependencies. Dies ist der Eingabeordner für jPackage und wird zur Sicherheit zu Beginn des Skriptes vor dem Maven-Build geleert.
- Die gezeigte jar-Datei *testmodul-0.0.1.jar* kann mit dem Befehl von Folie 16 ausgeführt werden. Es wäre also möglich die jar-Datei mit dem Ordner lib weiterzugeben. Es ist aber zu beachten, dass die Dependencies in *lib* plattformabhängig sind.



Installer erzeugen mit jPackage – Das Ergebnis

- Im Ordner *installers* gibt es nun einen Ordner, der nach der im Skript eingestellten Versionsnummer benannt ist.

  Dieser enthält einen Unterordner für das Betriebssystem, in dem sich der/die von jPackage erzeugten Installer-Dateien befinden.
- Diese Installer können nun an die Endnutzer:innen weitergeben und plattformabhängig installiert werden



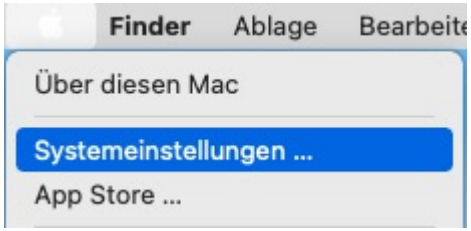
Installer erzeugen mit jPackage – Installation

- Beim Ausführen der Installer auf Windows und macOS können den Nutzer:innen Warnungen angezeigt werden, da das Programm nicht von Microsoft bzw. Apple verifiziert ist
  - Auf Windows genügt es die Meldung zu bestätigen und die Installation trotzdem auszuführen
  - Die nötigen Schritte für macOS werden auf den nächsten Folien gezeigt
- Bei Linux kann die graphische Installation unter Umständen fehlschlagen (in bisherigen Tests besonders bei rpm)
  - Das Paket kann aber trotzdem über das Terminal installiert werden
  - Der nötige Befehl ist je nach Distribution verschieden
  - Es empfiehlt sich daher auch statt oder zusätzlich zu den rpm/deb-Installern das sogenannte App-Image weiterzugeben. Dieses enthält Grundlegend die selben Bestandteile, die von den Installern in den Installationpfad kopiert werden. Die darin enthaltene ausführbare Datei im Unterverzeichnis *bin* kann auf allen Linux-Distributionen ausgeführt werden, wenn die nötigen Pakete installiert sind.
  - **Achtung:** Für den Build lieber ein älteres Linux mit älteren Paketen nutzen, da sonst die Versionsnummern der nötigen Pakete für LTS-Distributionen mit „älteren“ Paketquellen zu hoch sein können und das Programm dann nicht ausgeführt werden kann. Eine höhere installierte Version der nötigen Pakete auf dem Zielsystem ist dahingegen meist unproblematisch.

Installer erzeugen mit jPackage – Installation

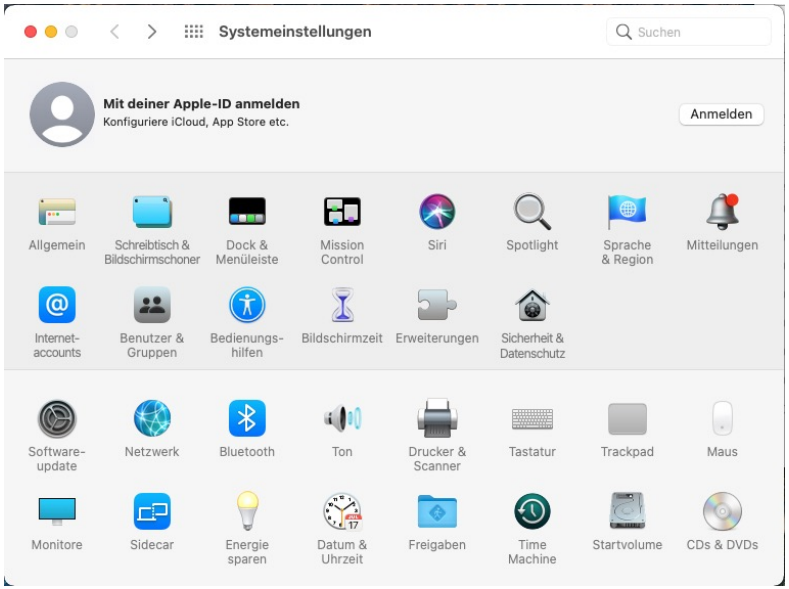
- Auf macOS wird beim ersten Öffnen des Installers diese Meldung angezeigt:

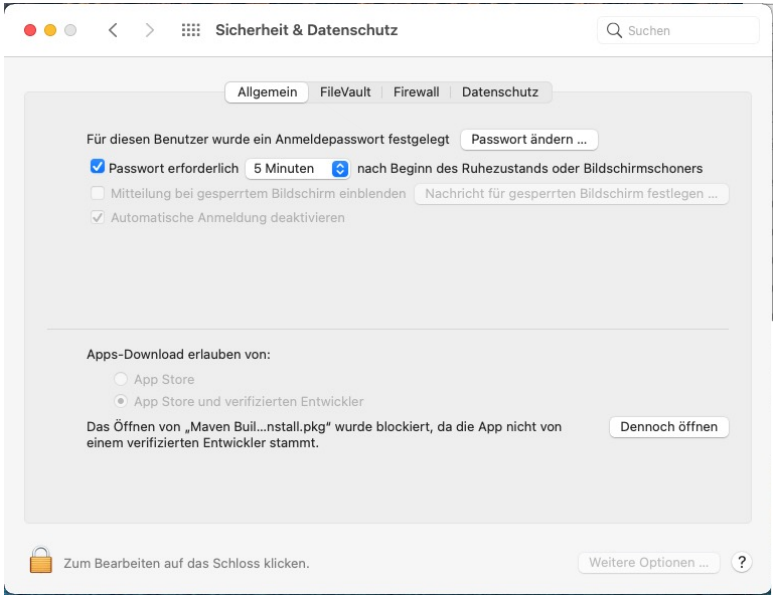
- Hier muss mit OK bestätigt werden
- Danach müssen oben links im Apple-Menü die Systemeinstellungen geöffnet werden



Installer erzeugen mit jPackage – Installation

- In den Einstellungen muss der Punkt „*Sicherheit und Datenschutz*“ ausgewählt werden
- Ggf. muss der Tab *Allgemein* angewählt





- Unten gibt es die Meldung, dass der Installer blockiert wurde
- Mit einem Klick auf „*Dennoch öffnen*“ neben der Meldung kann der Installer trotzdem ausgeführt werden

Installer erzeugen mit jPackage – Installation

- Zum Schluss wird eine letzte Warnmeldung angezeigt, die mit öffnen bestätigt werden muss

- **Hinweis:** Auf dem System, mit dem der Installer erzeugt wurde, sind diese Schritte meistens nicht notwendig, da das System wiedererkennt, dass es selber diesen Installer erzeugt hat

Quellen und weiterführende Links

- Zu Jar-Archiven:

  https://openbook.rheinwerk-verlag.de/javainsel9/javainsel_26_003.htm#mjf3bbdf106879ed579fef84173ca9be20

- Zu Maven:

  http://home.edvsz.fh-osnabrueck.de/skleuker/CSI/Werkzeuge/Maven/

  https://www.youtube.com/watch?v=Wm4OHUw954o&list=PLxXRtbg-a3UoZESWi1qwQieWfa5y4tsUl&index=2

- Zu jpackage:

  https://docs.oracle.com/en/java/javase/16/docs/specs/man/jpackage.html

  https://github.com/dlemmermann/JPackageScriptFX