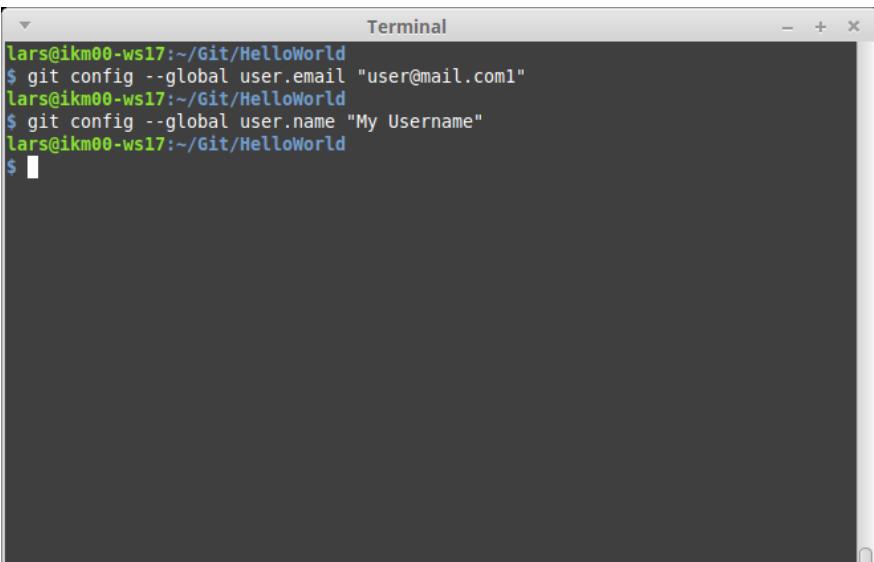


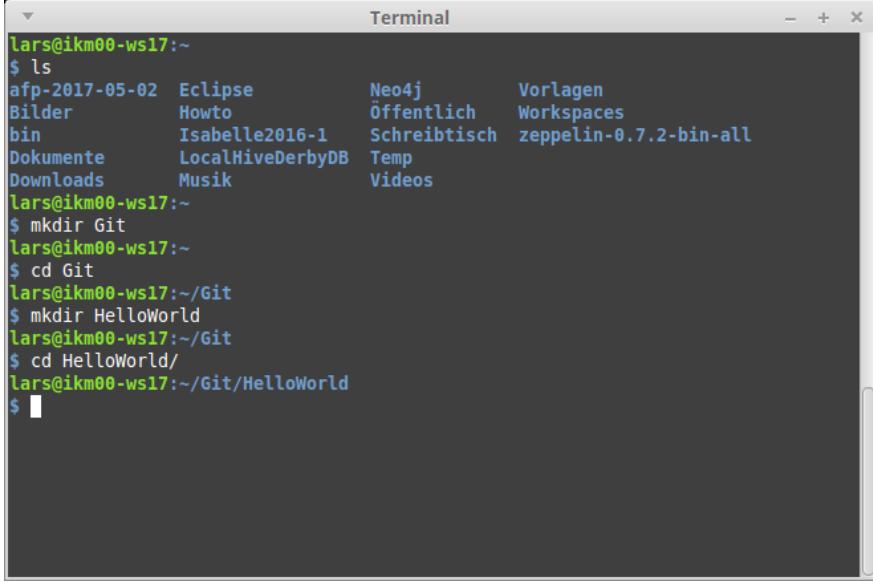
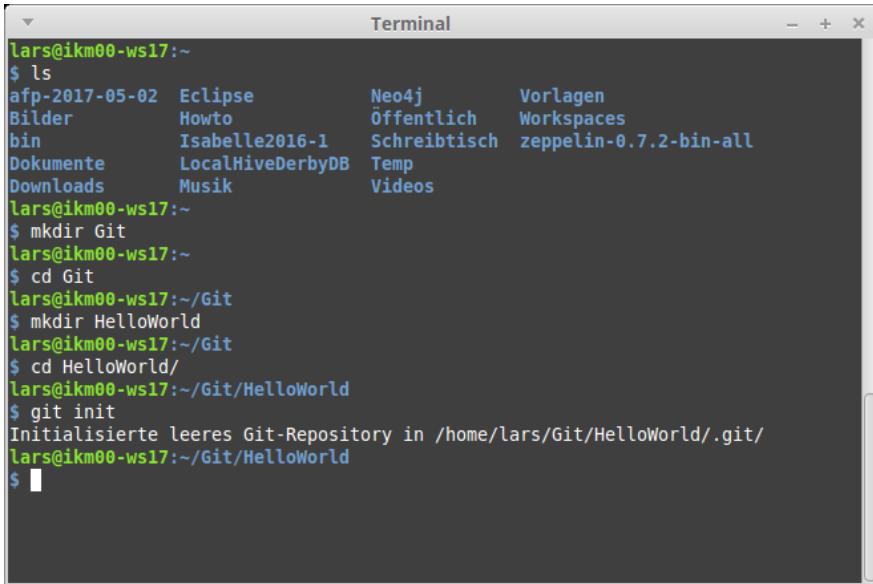
TUTORIAL GIT, PERISTENZ, MAVEN, JUNIT TESTS

TEIL 1: EINRICHTUNG VON GIT

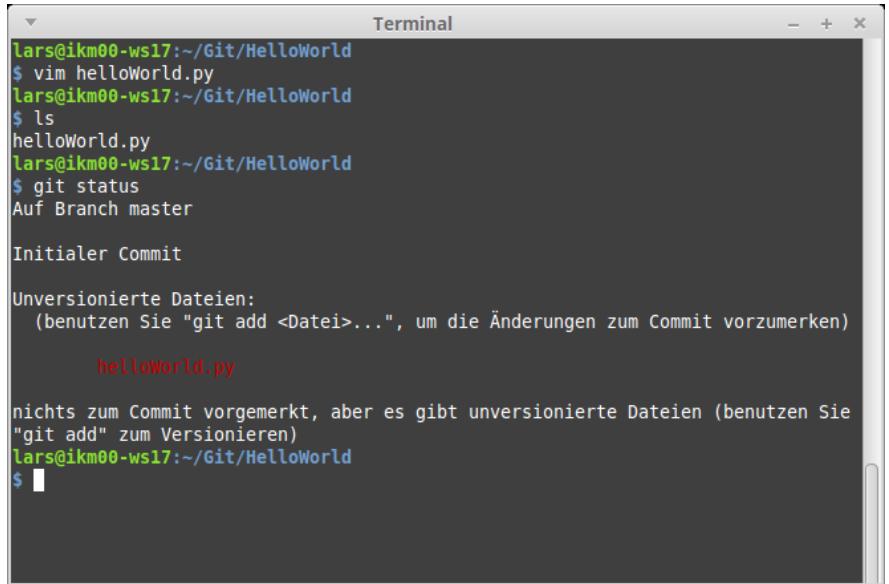
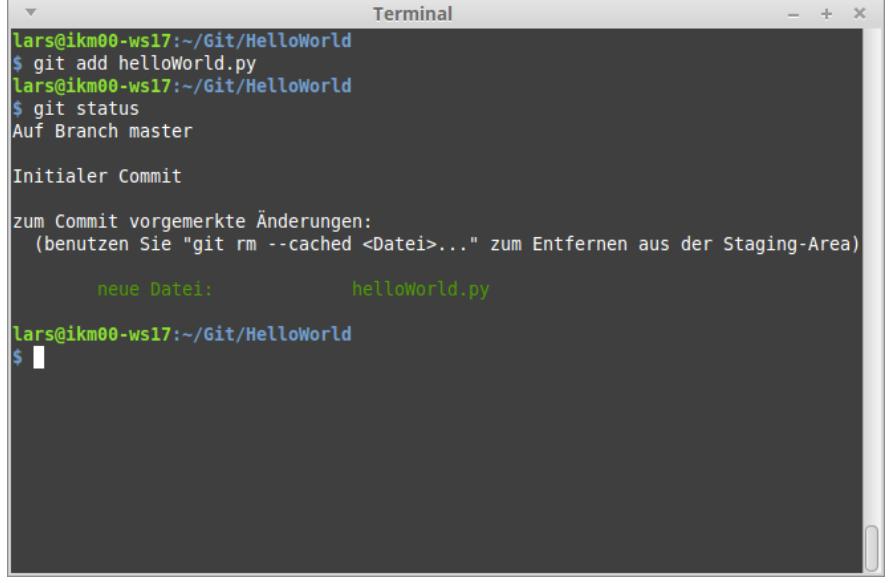
Explanation	Screenshot
<p>1. Bei der ersten Verwendung von Git muss zunächst ein Benutzer mit Namen und E-Mail eingetragen werden. Falls ein Benutzer in mehreren Repositorys wiederverwendet werden soll, kann ein globaler Benutzer per --global eingetragen werden. Der Kommandozeilenbefehl lautet:</p> <pre>git config user.name <username></pre> <p>bzw.</p> <pre>git config user.email <email></pre>	 <p>The screenshot shows a terminal window titled "Terminal". The command history is as follows:</p> <pre>lars@ikm00-ws17:~/Git/HelloWorld \$ git config --global user.email "user@mail.com1" lars@ikm00-ws17:~/Git/HelloWorld \$ git config --global user.name "My Username" lars@ikm00-ws17:~/Git/HelloWorld \$</pre>

TEIL 2: REPOSITORY ERSTELLEN

Falls Sie bereits Erfahrungen mit Git haben können Sie zu Teil 7 springen. In den folgenden Teilen werden die grundsätzlichen Git Kommandos wiederholt.

Explanation	Screenshot
1. In das Verzeichnis navigieren, in dem das Repository angelegt werden soll.	 A terminal window titled 'Terminal' showing the following command sequence: <pre>lars@ikm00-ws17:~\$ ls afp-2017-05-02 Eclipse Neo4j Vorlagen Bilder Howto Öffentlich Workspaces bin Isabelle2016-1 Schreibtisch zeppelin-0.7.2-bin-all Dokumente LocalHiveDerbyDB Temp Downloads Musik Videos</pre> <pre>lars@ikm00-ws17:~\$ mkdir Git lars@ikm00-ws17:~\$ cd Git lars@ikm00-ws17:~/Git\$ mkdir HelloWorld lars@ikm00-ws17:~/Git\$ cd HelloWorld/ lars@ikm00-ws17:~/Git/HelloWorld\$</pre>
2. Git Repository initialisieren:	 A terminal window titled 'Terminal' showing the following command sequence: <pre>lars@ikm00-ws17:~\$ ls afp-2017-05-02 Eclipse Neo4j Vorlagen Bilder Howto Öffentlich Workspaces bin Isabelle2016-1 Schreibtisch zeppelin-0.7.2-bin-all Dokumente LocalHiveDerbyDB Temp Downloads Musik Videos</pre> <pre>lars@ikm00-ws17:~\$ mkdir Git lars@ikm00-ws17:~\$ cd Git lars@ikm00-ws17:~/Git\$ mkdir HelloWorld lars@ikm00-ws17:~/Git\$ cd HelloWorld/ lars@ikm00-ws17:~/Git/HelloWorld\$ git init Initialisierte leeres Git-Repository in /home/lars/Git/HelloWorld/.git/ lars@ikm00-ws17:~/Git/HelloWorld\$</pre>
3. Das Repository ist einsatzbereit!	

TEIL 3: STAGING AND COMMITTING

Explanation	Screenshot
<p>1. Sobald Sie mit den Änderungen in Ihrem Arbeitsverzeichniss zufrieden sind können Sie eine Liste aller geänderten Dateien anzeigen lassen:</p> <pre>git status</pre>	 <pre>lars@ikm00-ws17:~/Git/HelloWorld \$ vim helloWorld.py lars@ikm00-ws17:~/Git/HelloWorld \$ ls helloWorld.py lars@ikm00-ws17:~/Git/HelloWorld \$ git status Auf Branch master Initialer Commit Unversionierte Dateien: (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken) helloWorld.py nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien (benutzen Sie "git add" zum Versionieren) lars@ikm00-ws17:~/Git/HelloWorld \$</pre>
<p>2. Nachdem eine neue Datei erstellt wurde ist sie zunächst „untracked“. Damit sie von Git verwaltet werden kann muss sie mit <code>git add <dateiname></code> mit aufgenommen werden. („Staging“).</p> <p>3. Nun zeigt <code>git status</code> einen neuen Zustand: „zum Commit vorgemerkte Änderungen“. Alle Änderungen der hier gelisteten Dateien werden beim nächsten Commit abgespeichert.</p>	 <pre>lars@ikm00-ws17:~/Git/HelloWorld \$ git add helloWorld.py lars@ikm00-ws17:~/Git/HelloWorld \$ git status Auf Branch master Initialer Commit zum Commit vorgemerkte Änderungen: (benutzen Sie "git rm --cached <Datei>..." zum Entfernen aus der Staging-Area) neue Datei: helloWorld.py lars@ikm00-ws17:~/Git/HelloWorld \$</pre>

4. Ein Commit wird mit dem Befehl `git commit -m <message>` durchgeführt. Dadurch werden die Änderungen der „staged“ Dateien abgespeichert.

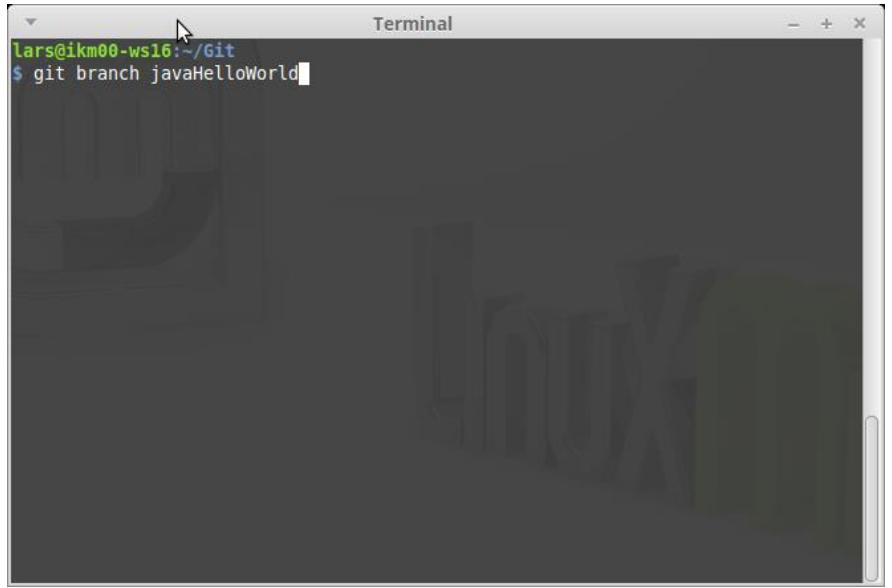
```
Terminal
lars@ikm00-ws17:~/Git/HelloWorld
$ git commit -m "Erstellung neues HelloWorld Programms in Python"
[master (Basis-Commit) d47df18] Erstellung neues HelloWorld Programms in Python
 1 file changed, 3 insertions(+)
 create mode 100644 helloWorld.py
lars@ikm00-ws17:~/Git/HelloWorld
$
```

5. Falls alle Änderungen gestaged wurden, zeigt `git status` nach dem commit an, dass das repo clean ist und keine Änderungen vorliegen

```
Terminal
lars@ikm00-ws17:~/Git/HelloWorld
$ git status
Auf Branch master
nichts zu committen, Arbeitsverzeichnis unverändert
lars@ikm00-ws17:~/Git/HelloWorld
$
```

6. Dieses Vorgehen wird für jeden Commit wiederholt.

TEIL 4: ARBEITEN MIT BRANCHES

Explanation	Screenshot
<p>1. Einen neuen Branch erstellen:</p> <pre data-bbox="192 406 541 440">git branch <branchname></pre>	 <p>A screenshot of a terminal window titled "Terminal". The window shows a command line with the user's name "lars" and the host "ikm00-ws16" followed by the command "\$ git branch javaHelloWorld". The cursor is positioned at the end of the command line.</p>
<p>2. In den neuen Branch wechseln:</p> <pre data-bbox="192 1028 382 1096">git checkout <branchname></pre>	 <p>A screenshot of a terminal window titled "Terminal". The window shows a command line with the user's name "lars" and the host "ikm00-ws16" followed by the command "\$ git checkout javaHelloWorld". The cursor is positioned at the end of the command line.</p>

3. Neue Änderungen machen und commiten

```
Terminal
lars@ikm00-ws16:~/Git
$ git checkout javaHelloWorld
Zu Branch 'javaHelloWorld' gewechselt
lars@ikm00-ws16:~/Git
$ vim javaHelloWorld.java
lars@ikm00-ws16:~/Git
$ git add javaHelloWorld.java
lars@ikm00-ws16:~/Git
$ git commit -m "Added java HelloWorld"
[javaHelloWorld ebbcfab] Added java HelloWorld
1 file changed, 5 insertions(+)
create mode 100644 javaHelloWorld.java
lars@ikm00-ws16:~/Git
$
```

4. In den master Branch zurückwechseln

```
Terminal
lars@ikm00-ws16:~/Git
$ git checkout master
Zu Branch 'master' gewechselt
lars@ikm00-ws16:~/Git
$
```

5. Den neuen Branch in den aktuellen mergen:

```
git merge <branchname>
```

```
Terminal
lars@ikm00-ws16:~/Git
$ git merge javaHelloWorld
Aktualisiere 8143127..ebbcfab
Fast-forward
 javaHelloWorld.java | 5 +++
 1 file changed, 5 insertions(+)
 create mode 100644 javaHelloWorld.java
lars@ikm00-ws16:~/Git
$
```

Save dialog box:

- Name: Bildschirmfoto-Terminal 3.png
- Im Ordner speichern: egit
- Hilfe
- In Zwischenablage kopieren
- Abbrechen
- Speichern

-
6. Alle Änderung, die im Branch
javaHelloWorld gemacht
wurden, sind jetzt auch in
master.
-

TEIL 5: GITHUB ALS REMOTE FÜR EIN NEUES PROJEKT

Explanation	Screenshot
<p>1. Anmeldung auf https://github.com/login</p> <p>Hinweis: Falls benötigt oder erwünscht ist die Erstellung eines neuen Accounts unter dem Link „Create an account“ möglich. Standard Accounts auf GitHub sind umsonst, können aber nur öffentliche Repositorys erstellen</p>	
2. Ein neues Repository anlegen	
3. Den Namen des Repositorys eintragen und „Create Repository“ klicken.	

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



THI-Tutorial-User

Repository name



RemoteTutorial



Great repository names are short and memorable. Need inspiration? How about [automatic-engine](#).

Description (optional)

Public

Anyone can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None**

Add a license: **None**



Create repository

4. Ein neues Projekt anlegen und ein git Repository mit git init initialisieren. In diesem Projekt die Readme anlegen:

```
echo "# RemoteTutorial"
>> README.md
```

Terminal

```
lars@ikm00-ws16:~/git
$ cd RemoteTutorial/
lars@ikm00-ws16:~/git/RemoteTutorial$ git init
Initialisierte leeres Git-Repository in /home/lars/git/RemoteTutorial/.git/
lars@ikm00-ws16:~/git/RemoteTutorial$ echo "# RemoteTutorial" >> README.md
lars@ikm00-ws16:~/git/RemoteTutorial$
```

new repository on the command line

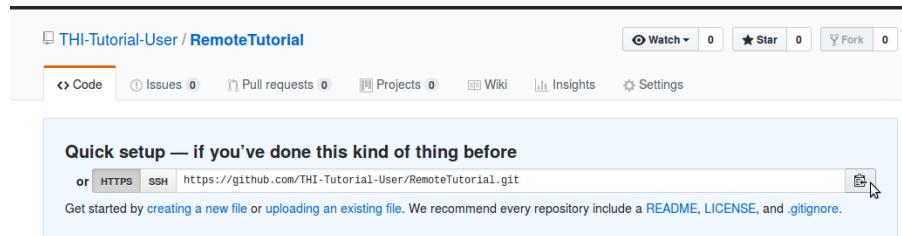
```
lars@ikm00-ws16:~/git/RemoteTutorial$ git add README.md
lars@ikm00-ws16:~/git/RemoteTutorial$ git commit -m "first commit"
lars@ikm00-ws16:~/git/RemoteTutorial$ git push -u origin master
```

listing repository from the command line

5. Die Readme committen

```
Terminal
lars@ikm00-ws16:~/git/RemoteTutorial
$ git add README.md
lars@ikm00-ws16:~/git/RemoteTutorial
$ git commit -m "first commit" README, LICENSE, and .gitignore.
[master (Basis-Commit) 0b4dd05] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
lars@ikm00-ws16:~/git/RemoteTutorial
$ 
lars@ikm00-ws16:~/git/RemoteTutorial
$ 
```

6. Die URL des GitHub Repository kopieren



THI-Tutorial-User / RemoteTutorial

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Quick setup — if you've done this kind of thing before

or [HTTPS](https://github.com/THI-Tutorial-User/RemoteTutorial.git) [SSH](https://github.com/THI-Tutorial-User/RemoteTutorial.git) <https://github.com/THI-Tutorial-User/RemoteTutorial.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

7. Das GitHub Repository als Remote hinzufügen:

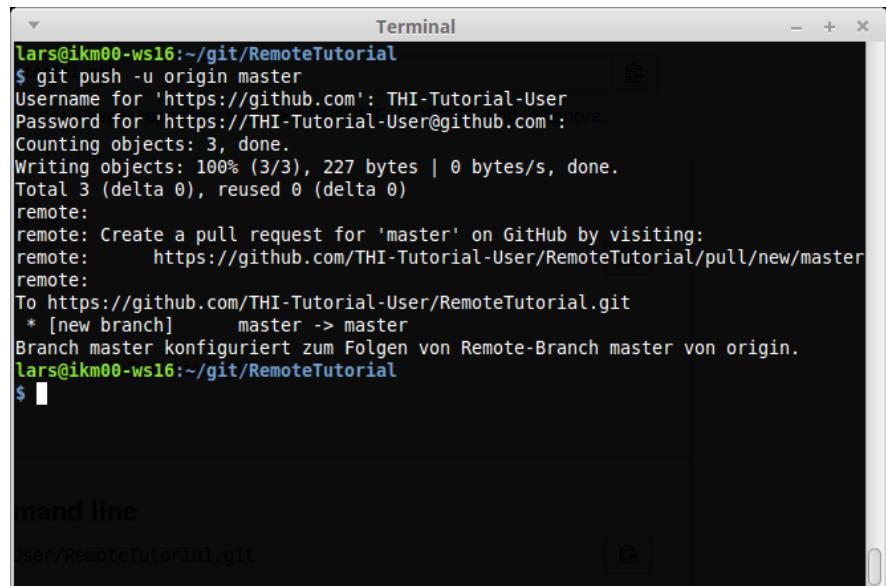
*git remote add origin
<URL>*

```
Terminal
lars@ikm00-ws16:~/git/RemoteTutorial
$ git remote add origin https://github.com/THI-Tutorial-User/RemoteTutorial.git
lars@ikm00-ws16:~/git/RemoteTutorial
$ 
```

8. Den Status des Projektes in das GitHub Repository hochladen:

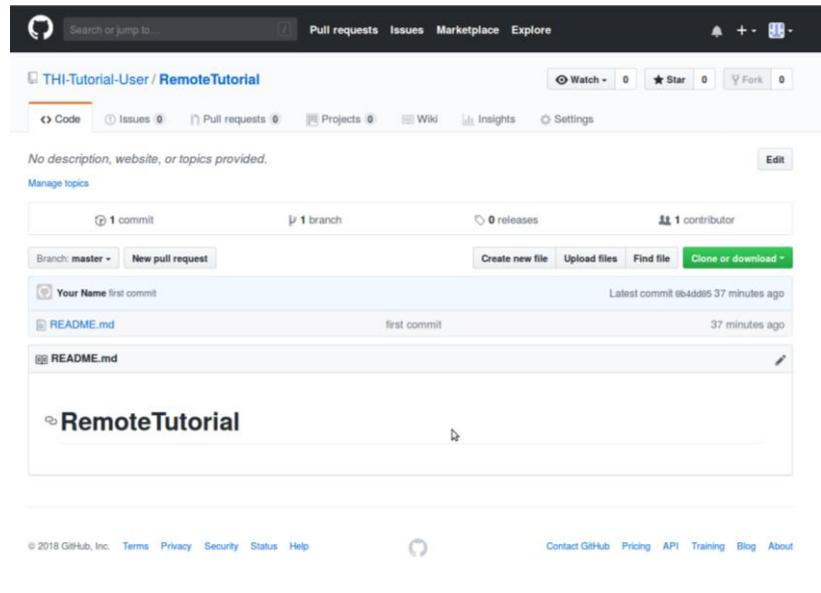
```
git push -u origin  
master
```

Wenn angefragt Benutzername und Passwort angeben.

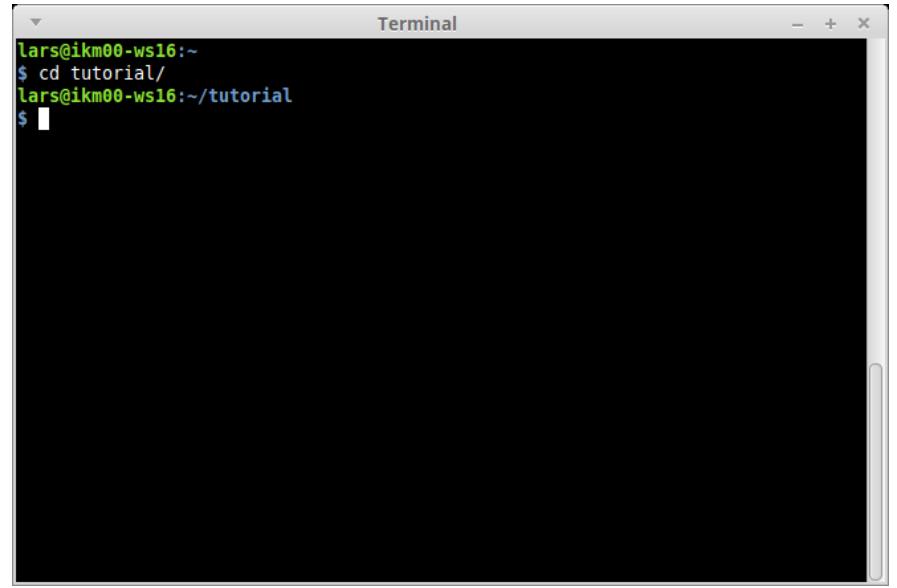
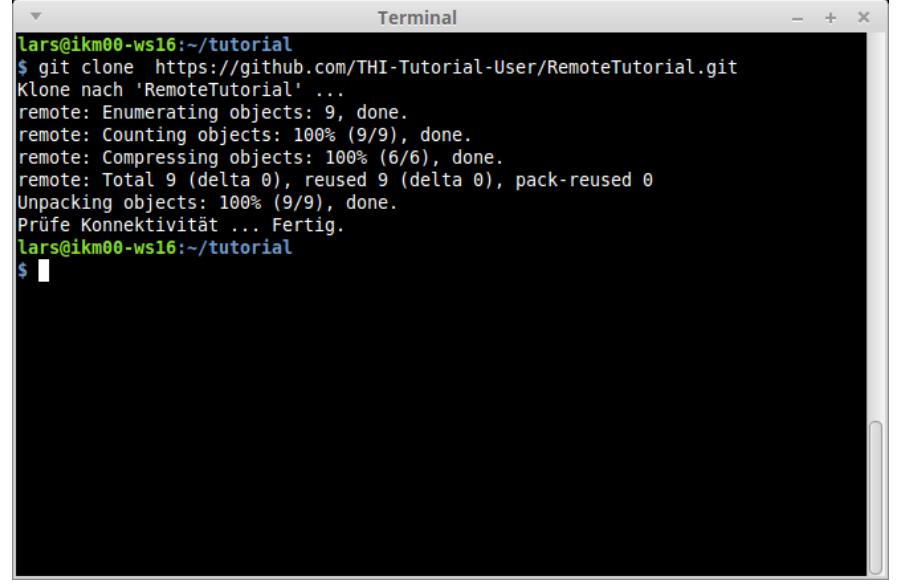


```
Terminal  
lars@ikm00-ws16:~/git/RemoteTutorial  
$ git push -u origin master  
Username for 'https://github.com': THI-Tutorial-User  
Password for 'https://THI-Tutorial-User@github.com':  
Counting objects: 3, done.  
Writing objects: 100% (3/3), 227 bytes | 0 bytes/s, done.  
Total 3 (delta 0), reused 0 (delta 0)  
remote:  
remote: Create a pull request for 'master' on GitHub by visiting:  
remote: https://github.com/THI-Tutorial-User/RemoteTutorial/pull/new/master  
remote:  
To https://github.com/THI-Tutorial-User/RemoteTutorial.git  
 * [new branch]      master -> master  
Branch master konfiguriert zum Folgen von Remote-Branch master von origin.  
lars@ikm00-ws16:~/git/RemoteTutorial  
$
```

9. Auf GitHub ist nun der aktuelle Status zu sehen



TEIL 6: EIN BESTEHENDES REPOSITORY CLONEN, REMOTE BRANCHES AUSCHECKEN UND MERGE-KONFLIKTE LÖSEN

Explanation	Screenshot
<p>1. Wechsle in ein Verzeichnis, in dem noch kein Git Repository existiert.</p>	 <p>Terminal window showing the command: <code>lars@ikm00-ws16:~\$ cd tutorial/</code> and the resulting directory path: <code>lars@ikm00-ws16:~/tutorial\$</code></p>
<p>2. Klonen des Tutorial Repository:</p> <pre>git clone https://github.com/THI-Tutorial-User/RemoteTutorial.git</pre>	 <p>Terminal window showing the command: <code>lars@ikm00-ws16:~/tutorial\$ git clone https://github.com/THI-Tutorial-User/RemoteTutorial.git</code> and the output of the cloning process, including object enumeration, compression, and unpacking.</p>

- ### 3. In das neue Verzeichnis wechseln:

```
cd RemoteTutorial
```

und ein erweitertes Log betrachten:

```
git log --graph --all -  
-decorate
```

```
lars@ikm00-ws16:~/tutorial/RemoteTutorial
$ git log --graph --all --decorate
* commit 811728c57e4f11a54309b0c7c22385a493eca5029 (HEAD, origin/master, origin/HEAD, master)
| Author: Your Name <you@example.com>
| Date:  Tue Nov 13 19:20:04 2018 +0100
|
|     Added more content to example text file
|
* commit e23c01bf5dc1cc780a555d264f7399f6437674a3 (origin/tutorialBranch)
| Author: Your Name <you@example.com>
| Date:  Tue Nov 13 19:18:01 2018 +0100
|
|     Added example file
|
* commit 0b4dd05a9915c3fd96e948bb3a22c4bce8494541
| Author: Your Name <you@example.com>
| Date:  Tue Nov 13 18:27:51 2018 +0100
|
|     first commit
lars@ikm00-ws16:~/tutorial/RemoteTutorial
$
```

4. Erstellen eines lokalen Branches, der dem Remote Branch *origin/tutorialBranch* folgt und den neuen Branch auschecken:

```
git checkout -b  
tutorialBranch  
origin/tutorialBranch
```

```
Terminal
lars@ikm00-ws16:~/tutorial/RemoteTutorial
$ git checkout -b tutorialBranch origin/tutorialBranch
Branch tutorialBranch konfiguriert zum Folgen von Remote-Branch tutorialBranch von origin.
Zu neuem Branch 'tutorialBranch' gewechselt
lars@ikm00-ws16:~/tutorial/RemoteTutorial
$
```

5. Die Datei `example.txt` mit neuem, beliebigen Text füllen und die Änderungen commiten

```
Terminal
lars@ikm00-ws16:~/tutorial/RemoteTutorial
$ git add example.txt
lars@ikm00-ws16:~/tutorial/RemoteTutorial
$ git commit -m "Added more text"
```

-
6. Zurück in den Branch *master* wechseln

```
Terminal
lars@ikm00-ws16:~/tutorial/RemoteTutorial
$ git checkout master
Zu Branch 'master' gewechselt
Ihr Branch ist auf dem selben Stand wie 'origin/master'.
lars@ikm00-ws16:~/tutorial/RemoteTutorial
$
```

-
7. Die Änderungen des Branches *tutorialBranch* in *master* zusammenführen:

```
git merge
tutorialBranch
```

Es entsteht ein merge-Konflikt.

```
Terminal
lars@ikm00-ws16:~/tutorial/RemoteTutorial
$ git merge tutorialBranch
automatischer Merge von example.txt
KONFLIKT (Inhalt): Merge-Konflikt in example.txt
Automatischer Merge fehlgeschlagen; beheben Sie die Konflikte und committen Sie dann das Ergebnis.
lars@ikm00-ws16:~/tutorial/RemoteTutorial
$
```

8. Die Datei *example.txt* mit einem Texteditor oder merge-Tool so bearbeiten, dass alle Änderungen behalten werden.

Hinweis:

Falls git zwei Versionen nicht automatisch zusammenfügen kann, muss man selbst Hand anlegen. Beide Versionen werden an den jeweiligen Stellen eingefügt, und markiert:

```
<<<<< HEAD
Don't remove this line
=====
Now there's more text
here
>>>>> tutorialBranch
```

Vor den „=” Zeichen steht der lokale Zustand der Datei, danach der des Branches der in den aktuellen gemergt werden soll. Man muss selbst entscheiden, welche der Versionen sinnvoll ist oder ob vielleicht auch beide eingefügt werden müssen.

example.txt (~/tutorial/RemoteTutorial) - VIM

```
1 Hello,
2
3 this is an example file
4
5 Don't remove this line
6 Now there's more text here
~
```

example.txt 1,1 Alles

9. Die gefixte Datei erneut stagern und commiten. In dem sich öffnenden Texteditor die Commit Nachricht anpassen und beschreiben, wie der merge-Konflikt gelöst wurde.

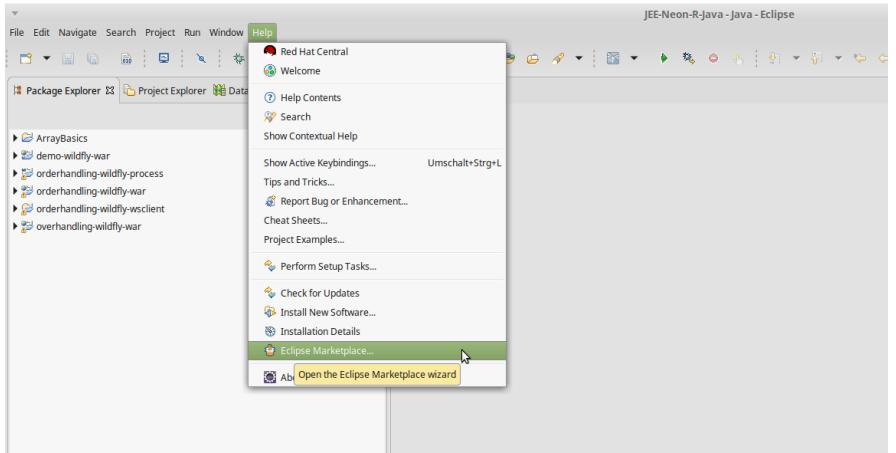
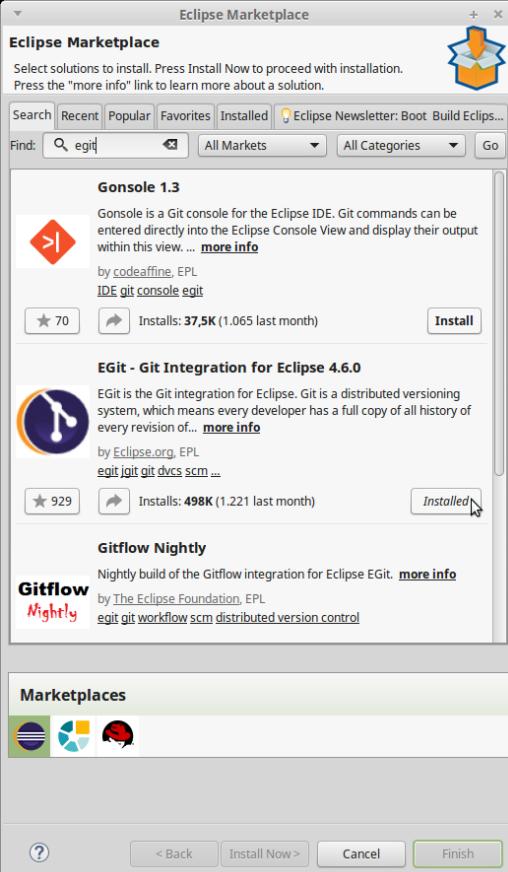
COMMIT_EDITMSG + (~/tutorial/RemoteTutorial/.git) - VIM

```
1 Merge branch 'tutorialBranch'
2
3 Conflicts:
4     example.txt Appended new changes to end of file
5 #
6 # Es sieht so aus, als committen Sie einen Merge.
7 # Falls das nicht korrekt ist, löschen Sie bitte die Datei
8 # .git/MERGE_HEAD
9 # und versuchen Sie es erneut.
10
11
12 # Bitte geben Sie eine Commit-Beschreibung für Ihre Änderungen ein. Zeilen,
13 # die mit '#' beginnen, werden ignoriert, und eine leere Beschreibung
14 # bricht den Commit ab.
15 # Auf Branch master
16 # Ihr Branch ist auf dem selben Stand wie 'origin/master'.
17 #
18 # Alle Konflikte sind behoben, aber Sie sind immer noch beim Merge.
19 #
20 # zum Commit vorgemerkt Änderungen:
21 #     geändert: example.txt
22 #
23 # Unversionierte Dateien:
24 #     example.txt.orig
25 #
```

.git/COMMIT_EDITMSG [+] 4,48-55 Alles

TEIL 7: EGIT INSTALLATION ÜBERPRÜFEN

Egit ist ein Git Client, der direkt in Eclipse integriert ist. Es ist genauso möglich, alle folgenden git Aufgaben auch mit der Kommandozeile zu bearbeiten. Die Wahl zwischen git Clients und direkte Verwendung eines Terminals ist letztlich eine Geschmacksfrage. Neben Egit existieren auch andere Clients wie z.B. Gitkraken.

Explanation	Screenshot
<p>1. Egit ist in der Virtuellen Maschine bereits in Eclipse installiert. Um die Installation zu überprüfen kann der Marketplace geöffnet werden.</p>	
<p>2. Nach Egit suchen, das Plugin wird als installiert angezeigt. Falls es nicht installiert ist, wie in vorherigen Tutorials installieren.</p>	

TEIL 8: GITHUB REPOSITORY KLONEN UND ECLIPSE PROJEKT IMPORTIEREN

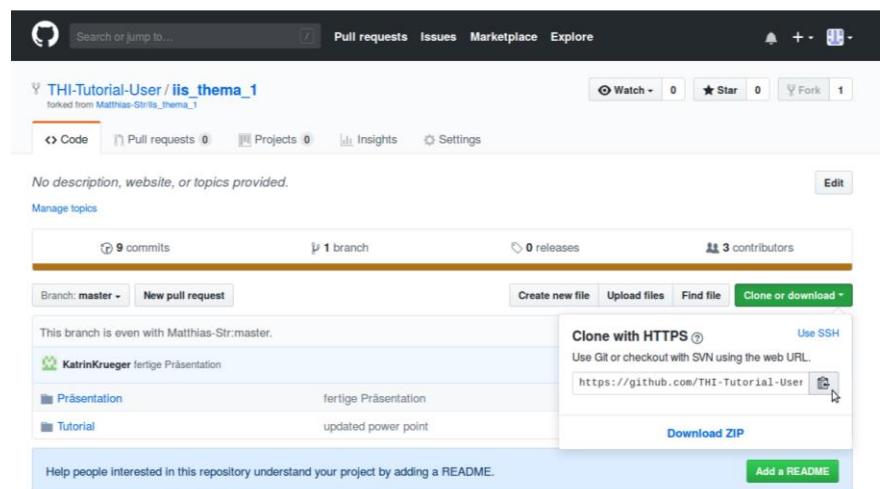
Im folgenden Abschnitt wird das erste Tutorial Repository auf GitHub geforkt und auf die lokale Maschine geklont. Danach wird mit EGit ein neuer Branch erstellt. Anschließend kann mit dem Persistenz Tutorial begonnen werden.

Falls ihr mit diesem Schritt bereits begonnen habt, das alte Repository löschen! Und auch das evtl. schon importierte Projekt in Eclipse wieder löschen.

Explanation	Screenshot
1. Öffnen des „Perspektive öffnen“ Dialogs	
2. Auswahl von „Git“	
3. Im Browser das Repository unter dem Link https://github.com/KatrinKrueger/iis_thema_1.git	

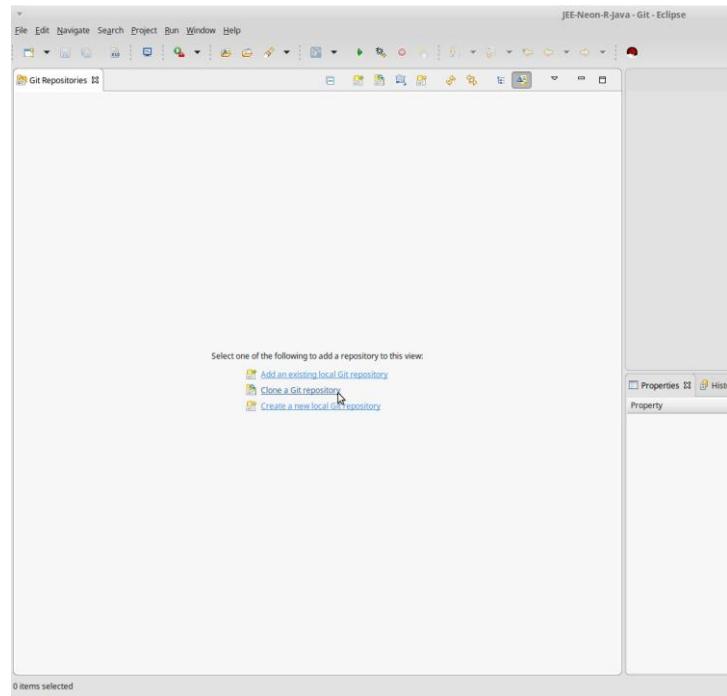
aufrufen. Mit dem Button Fork eine Kopie des Repositorys anlegen.

4. Die Repository URL kopieren



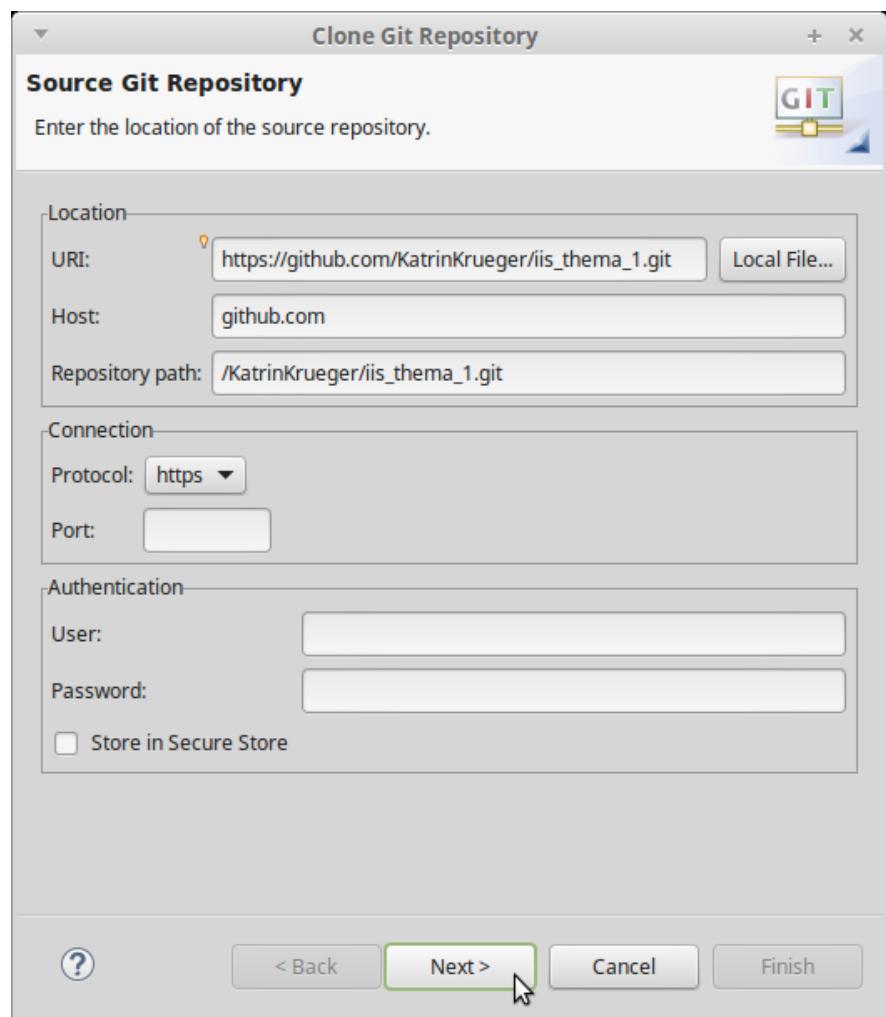
The screenshot shows a GitHub repository page for 'THI-Tutorial-User / iis_thema_1'. The page includes a search bar, navigation links for Pull requests, Issues, Marketplace, and Explore, and a header with a watch count of 0, a star count of 0, and a fork count of 1. Below the header, there are sections for 'Code', 'Pull requests', 'Projects', 'Insights', and 'Settings'. A message states 'No description, website, or topics provided.' and a 'Manage topics' button is available. Below this, a summary shows 9 commits, 1 branch, 0 releases, and 3 contributors. A 'Branch: master' dropdown and a 'New pull request' button are present. On the right, there are buttons for 'Create new file', 'Upload files', 'Find file', 'Clone or download', and 'Use SSH'. The 'Clone with HTTPS' button is highlighted with a mouse cursor, and the URL 'https://github.com/THI-Tutorial-User/iis_thema_1' is displayed. Other options include 'Download ZIP' and 'Add a README'.

5. Zurück in Eclipse „Clone a Git repository“ auswählen

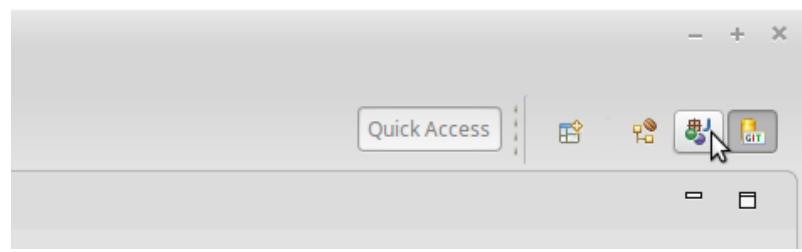


The screenshot shows the Eclipse Git interface with the title 'JEE-Neon-R-java - Git - Eclipse'. The menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations. The left panel is titled 'Git Repositories' and shows a list of repositories. The central workspace is empty. A message at the top of the workspace says 'Select one of the following to add a repository to this view:' with three options: 'Add an existing local Git repository', 'Clone a Git repository', and 'Create a new local Git repository'. The 'Clone a Git repository' option is highlighted with a mouse cursor. The right panel shows 'Properties' and 'History' tabs, with the 'Property' tab currently selected. The status bar at the bottom indicates '0 items selected'.

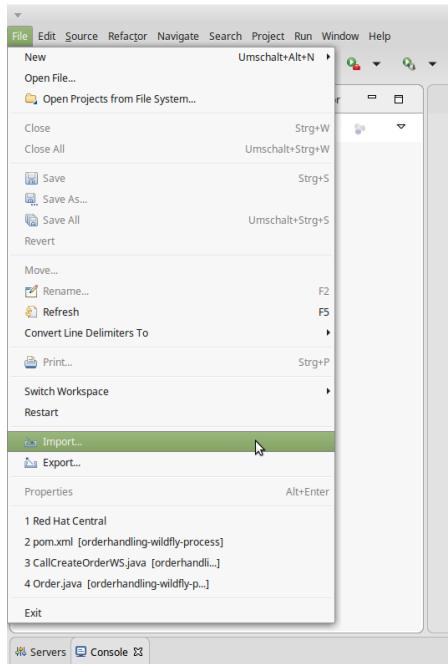
6. In dem Dialog die URI einfügen und auf finish klicken. Das Repository wird anschließend heruntergeladen.



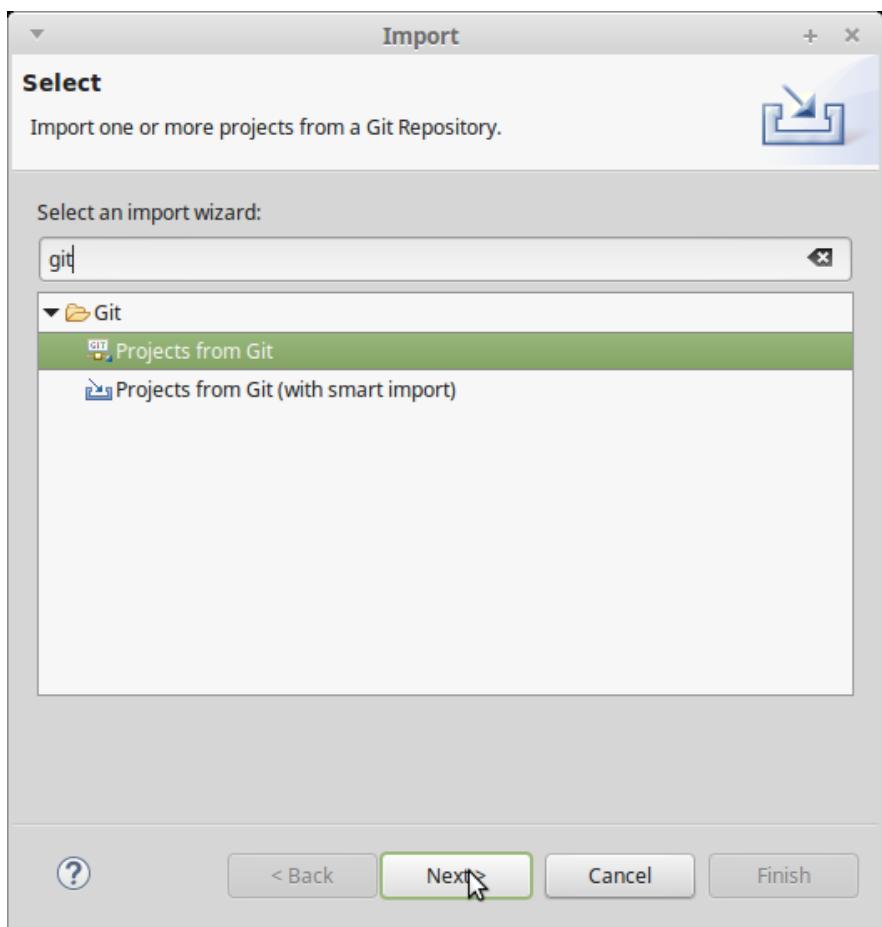
7. Wechseln in die Ansicht „Java“



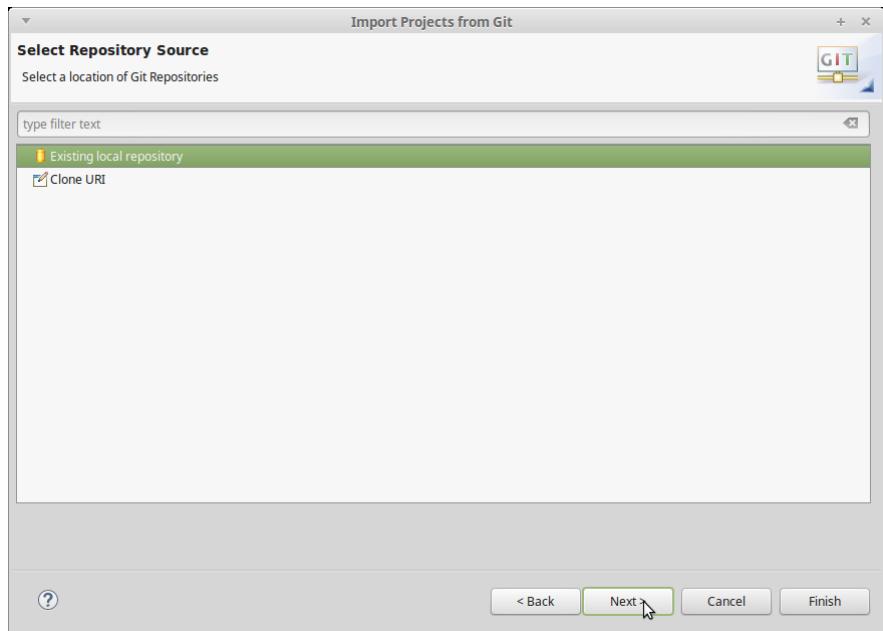
8. Im Dropdownmenü „File“ → „Import“ auswählen.



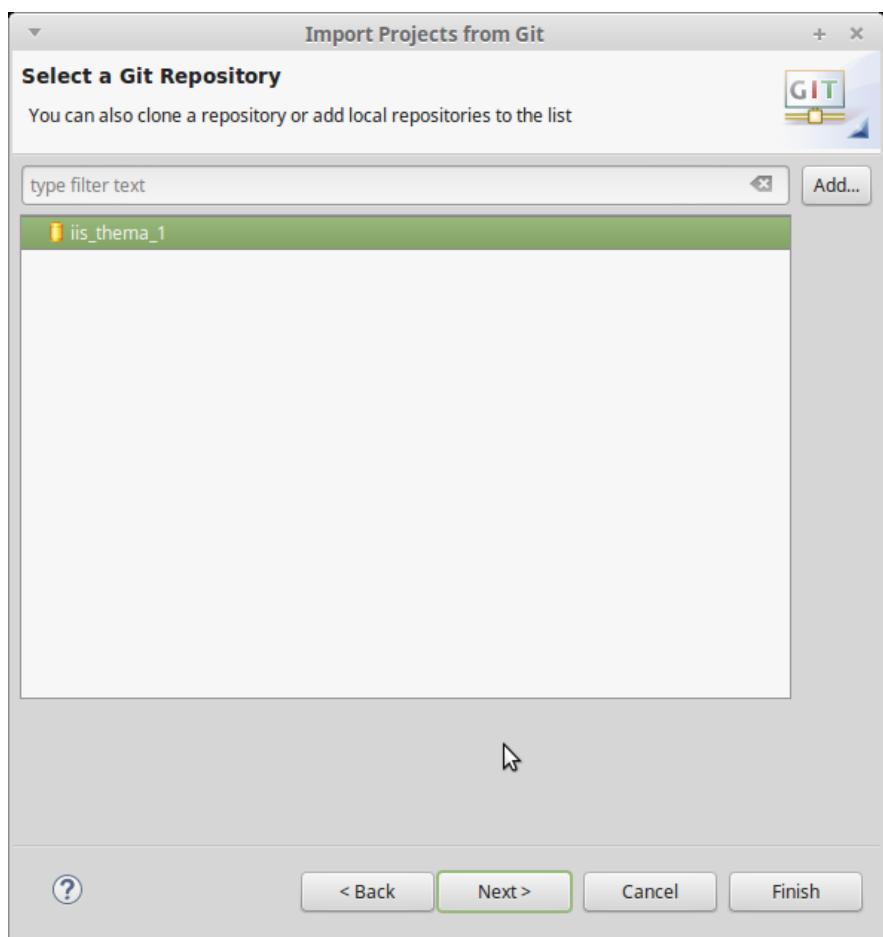
9. Nach git suchen, „Projects from Git“ markieren und auf Next klicken



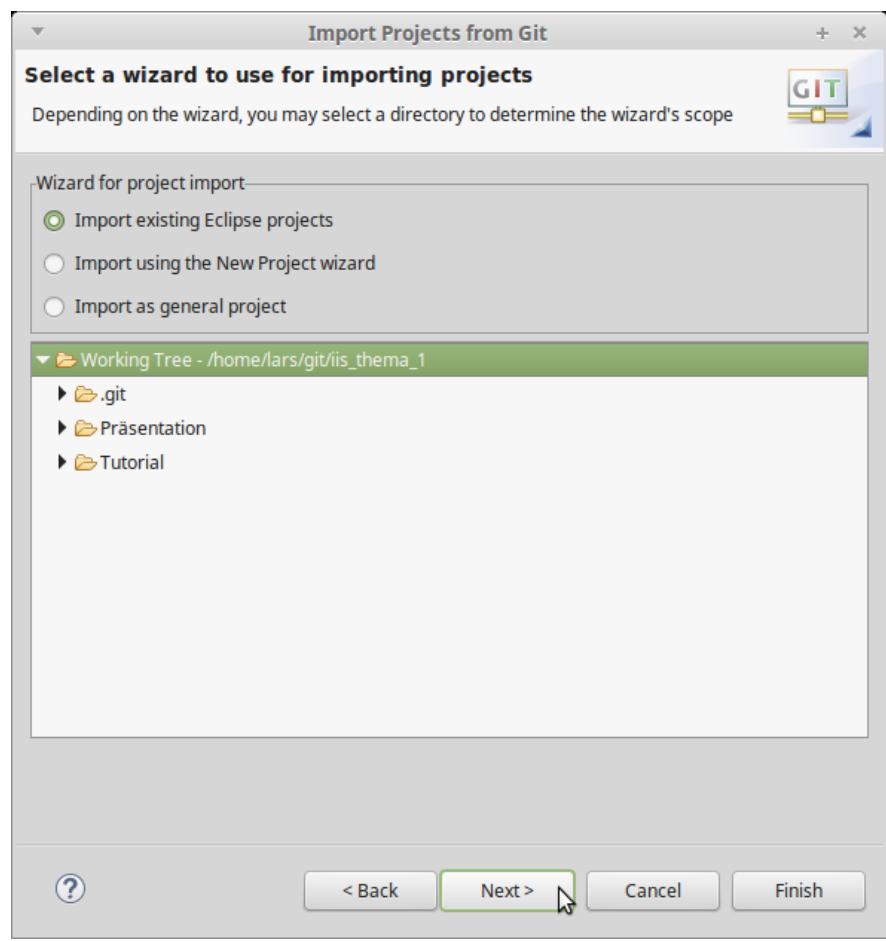
10. In nächsten Schritt „Existing local repository“ wählen und auf Next klicken.



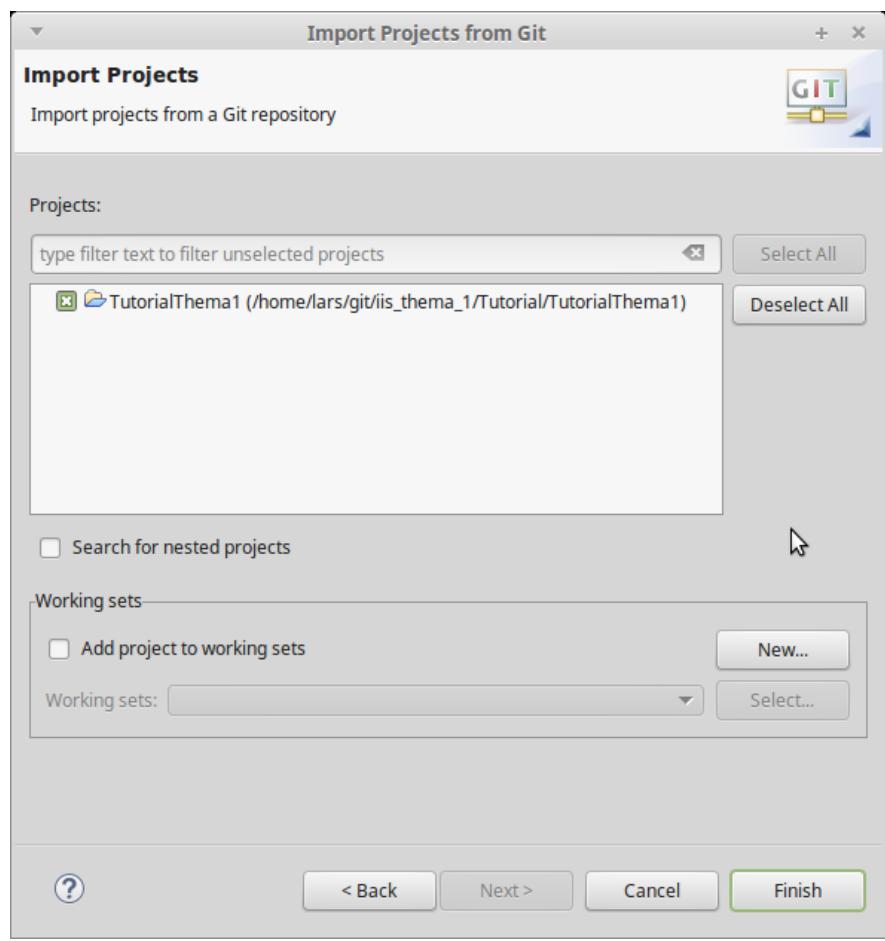
11. Das eben heruntergeladene Repository auswählen.



12. Die automatische Auswahl benutzen.



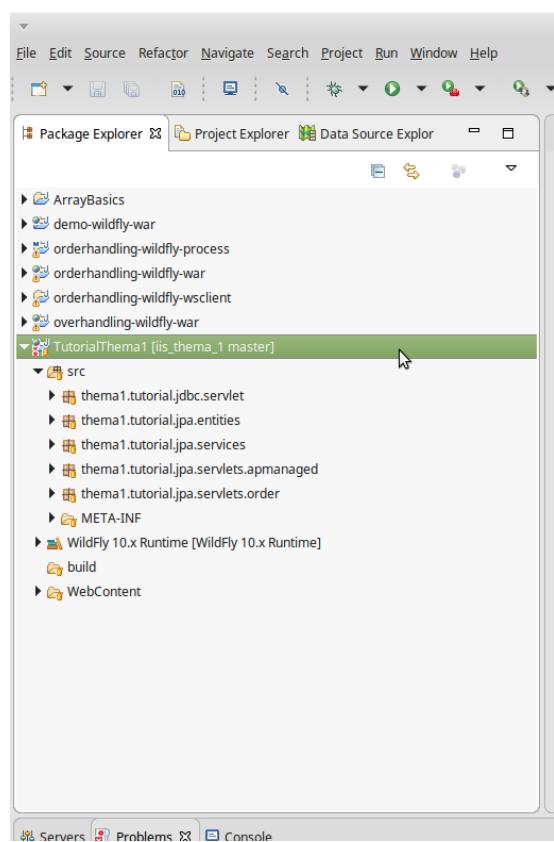
13. Eclipse zeigt die Liste der gefundenen Projekte, alle auswählen und den Dialog Finishen.



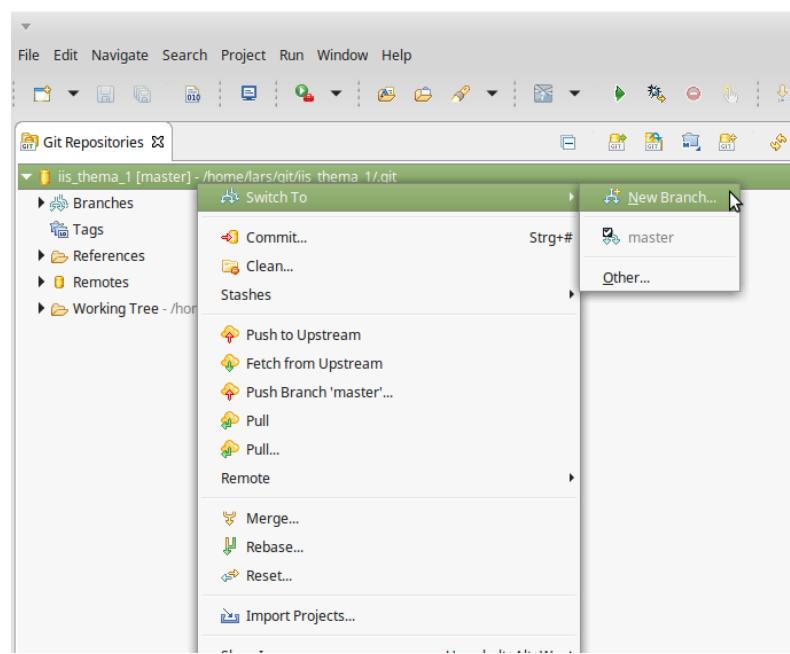
14. Das Projekt wird im Package Explorer angezeigt.

Sollten hier Fehler bei den Imports auftreten (evtl. auch das Projekt einmal builden), dann muss der java path angepassten werden. Dazu *Rechtsklick auf das Projekt > Properties*

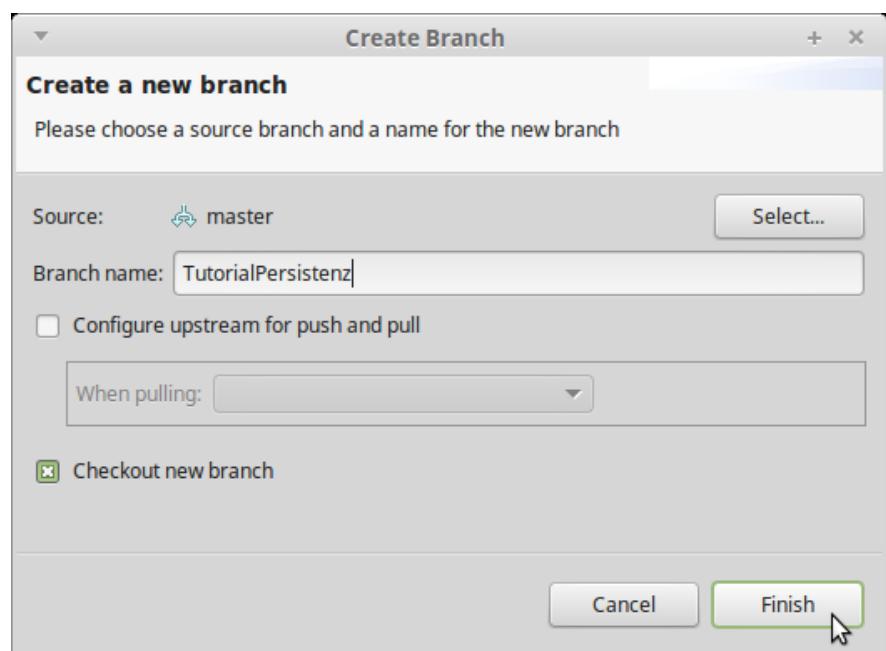
Im Fenster zu *Java Build Path* wechseln. Dort in den Reiter *Libraries* wechseln. Dort *JRE System Library* auswählen und auf *Edit* klicken. Bei *Alternate JRE* muss **jdk1.8.0_101** (das was wir in einem der ersten Tutorials eingerichtet haben) ausgewählt sein. Mit *Finish* bestätigen.



15. Wechseln zurück in die Ansicht Git. Rechtsklick auf das Repository und „Switch To -> New Branch“ auswählen.



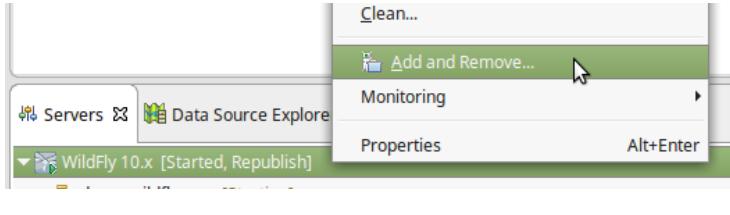
16. Den Namen TutorialPersistenz eingeben



17. Das Projekt ist nun bereit für das nächste Tutorial: Persistenz

TEIL 9: PERSISTENZ EINRICHTEN

Zunächst wird eine neue MySQL-Datenbank für das Tutorial angelegt. Anschließend wird die angelegte Datenbank als Ressource dem WildFly-Server hinzugefügt.

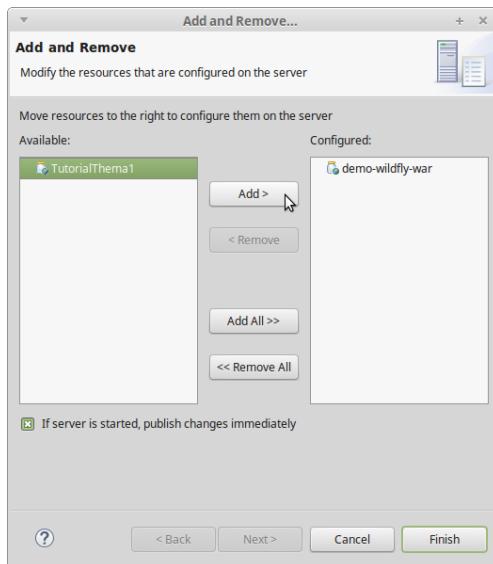
Explanation	Screenshot
<p>3. MySQL-Service starten:</p> <pre>sudo service mysql start Passwort: lars+glas</pre>	<pre>Lars@ikm00-ws16:~\$ sudo service mysql start * Starting MariaDB database server mysqld * Checking for corrupt, not cleanly closed and upgrade needing tables.</pre> <p>[OK]</p>
<p>4. MySQL -Konsole starten:</p> <pre>sudo mysql -u root -p Passwort: master42</pre>	<pre>Lars@ikm00-ws16:~\$ sudo mysql -u root -p Enter password: Welcome to the MariaDB monitor. Commands end with ; or \g. Your MariaDB connection id is 28 Server version: 5.5.49-MariaDB-1ubuntu0.14.04.1 (Ubuntu) Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others. Type 'help;' or '\h' for help. Type '\c' to clear the current input statement. MariaDB [(none)]></pre>
<p>5. Existierende Datenbanken anzeigen lassen:</p> <pre>show databases;</pre>	<pre>MariaDB [(none)]> show databases; +-----+ Database +-----+ information_schema hadoopguide mysql performance_schema +-----+ 4 rows in set (0.00 sec) MariaDB [(none)]></pre>
<p>6. Neues Schema für Tutorial erstellen:</p> <pre>create database shop;</pre>	<pre>MariaDB [(none)]> create database shop; Query OK, 1 row affected (0.00 sec)</pre>
<p>7. Erneut alle existierenden DB anzeigen lassen:</p> <pre>show databases;</pre> <p>Jetzt sollte die DB shop auch erscheinen.</p> <p>Hinweis: Tabellen müssen nicht händisch angelegt werden, da diese vom JPA-Projekt automatisch entsprechend der angelegten Entities erstellt werden.</p>	<pre>MariaDB [(none)]> show databases; +-----+ Database +-----+ information_schema hadoopguide mysql performance_schema shop +-----+</pre>
<p>8. Die erstellte Datenbank muss nun dem WildFly-Server als DataSource hinzugefügt werden. Dafür im Eclipse-Projekt: <i>Rechtsklick auf den WildFly-Server > Add and Remove...</i></p>	

9. Im sich öffnenden Fenster links das Projekt **TutorialThema1** auswählen und mit *Add* dem Server hinzufügen.

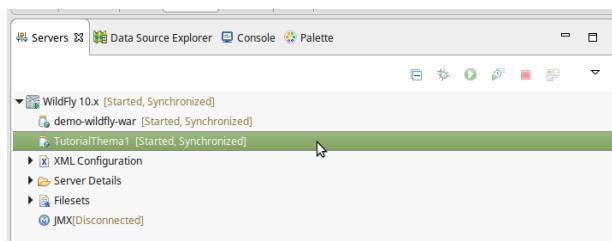
Dialog mit *Finish* beenden und schließen.

Hinweis:

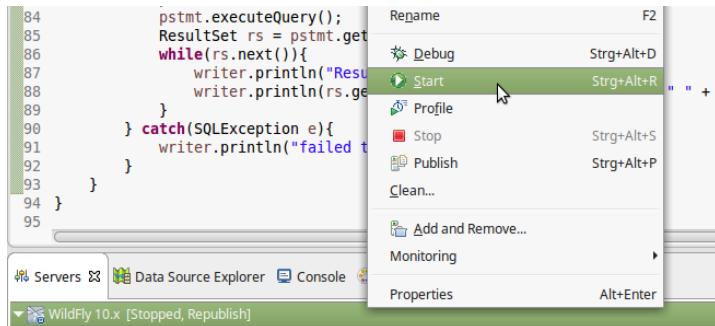
Damit eine eventuelle Fehlersuche leichter ist, am besten zuerst alle Projekte entfernen (*Remove All*) und dann nur das Tutorial-Projekt hinzufügen (*Add*).



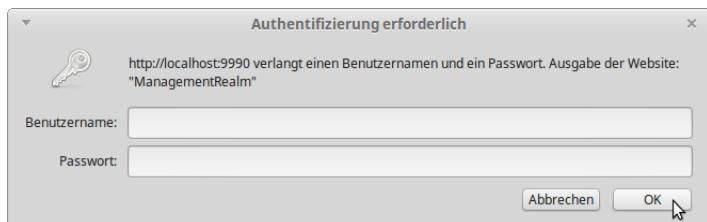
10. Klappt man nun den WildFly-Server auf, sollte dort das Projekt aufgelistet werden.



11. Nun den Server starten und im Browser die Adresse **localhost:9990** aufrufen. Das Starten dauert, und erst dann kann man die Adresse aufrufen.

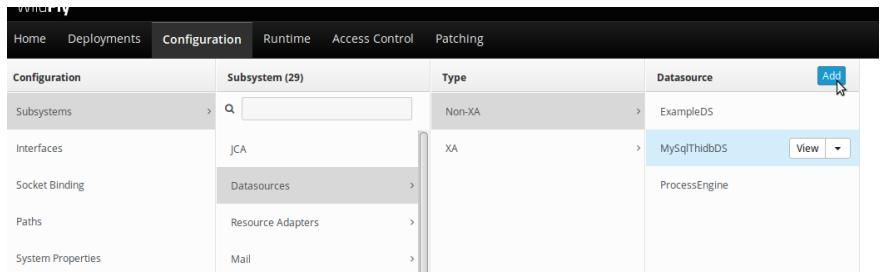


12. Anmelden mit
User: admin
Passwort: admin

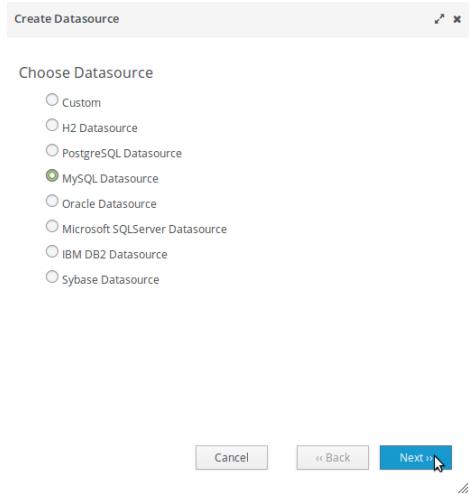


13. Im Reiter *Configuration* zu folgendem Pfad navigieren: *Subsystems > Datasources > Non-XA*

Anschließend den blauen Button *Add* klicken.

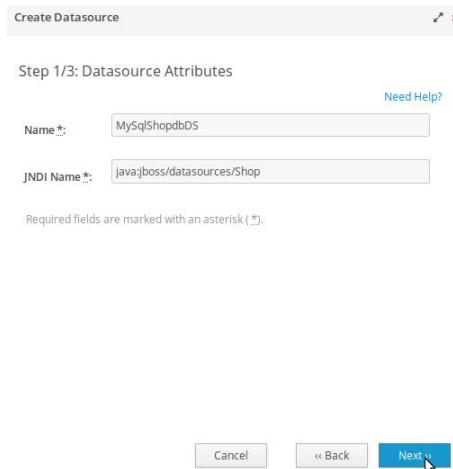


14. MySQL Datasource auswählen und mit *Next* weiter navigieren.

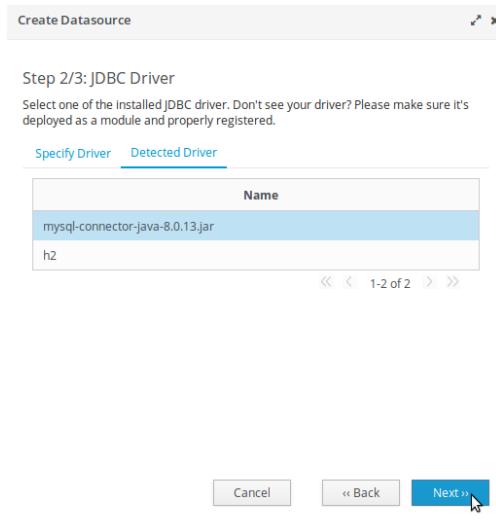


15. In Schritt 1/3 die Felder folgendermaßen befüllen:
Name: MySqlShopdbDS
JNDI Name:
java:jboss/datasources/
Shop

Weiter mit *Next*.

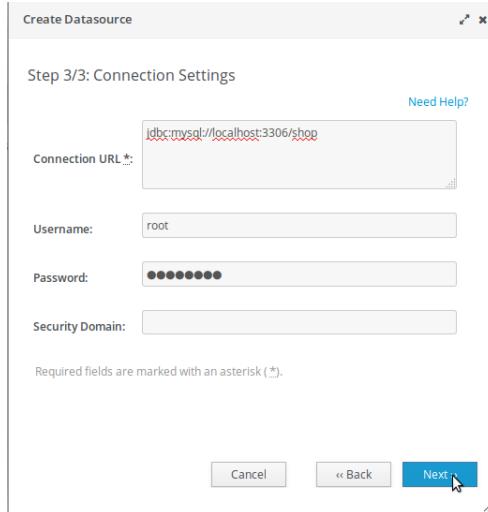


16. In Schritt 2/3 im Reiter *Detected Driver* den bereits hinterlegten mysql-Driver wählen und mit *Next* weiter navigieren.

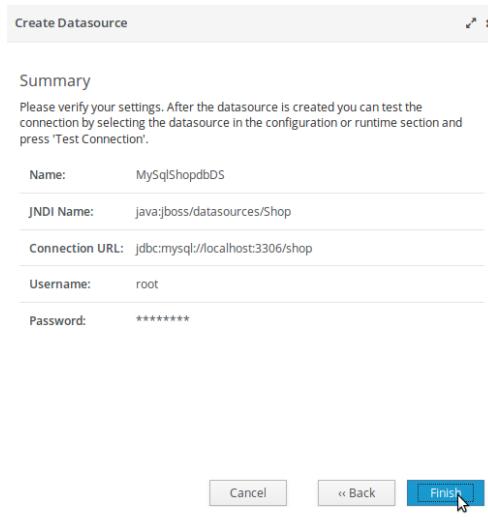


17. In Schritt 3/3 folgende Attribute eingeben:
Connection URL:
jdbc:mysql://localhost:3306/shop
Username: root
Passwort: master42

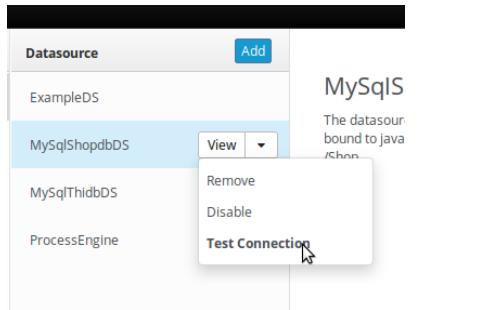
Mit *Next* weiter navigieren.



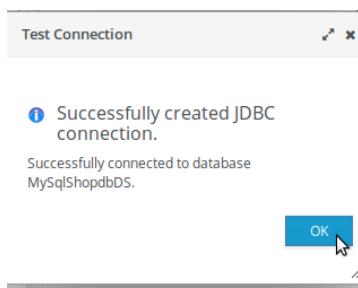
18. Eingegebene Daten im Summary überprüfen und mit *Finish* beenden.



19. Verbindung testen, indem im Dropdown neben der eben angelegten DataSource *Test Connection* ausgewählt wird.



20. Falls das Ergebnis so aussieht wie in nebenstehendem Bild, wurde die DataSource erfolgreich angelegt und kann im Folgenden genutzt werden.

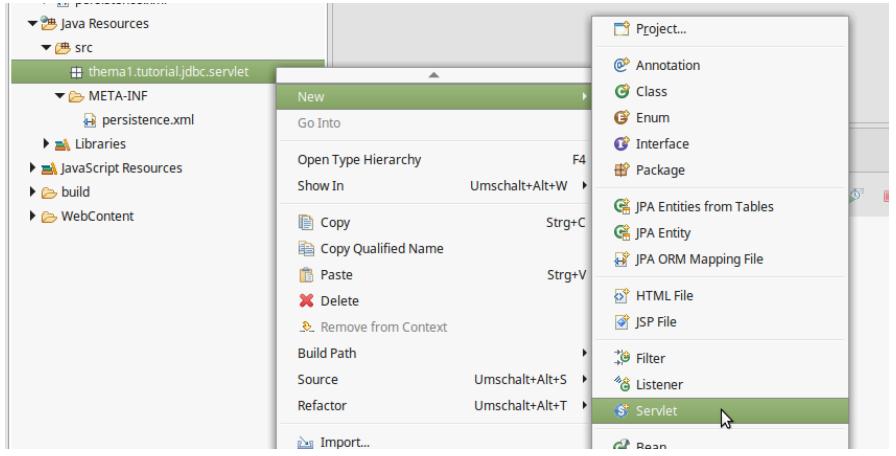
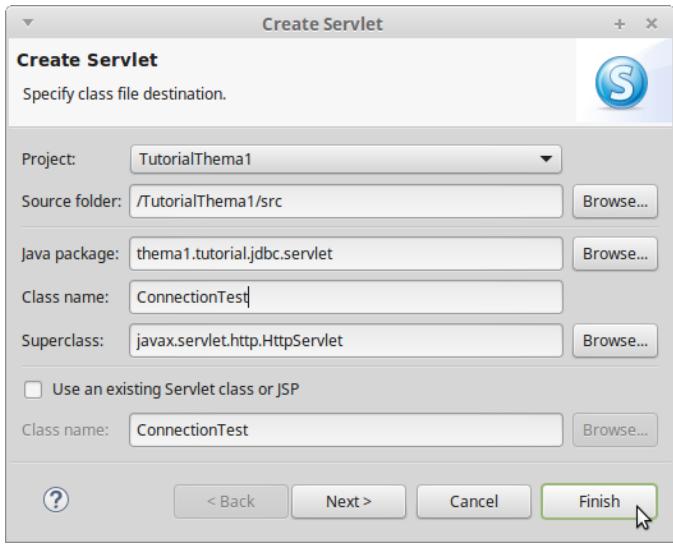


Sollte die Verbindung nicht erfolgreich aufgebaut werden können, alle eingegebenen Attribute überprüfen (keine nicht gewünschten Lehrszeichen, Buchstabendreher o.Ä.). Der MySQL-Server muss dafür gestartet sein (siehe Schritt 1).

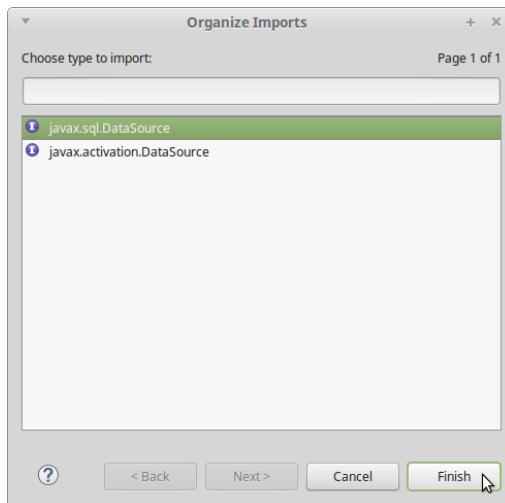
TEIL 10: JDBC

Im Folgenden wird ein neues Servlet erstellt, um die angelegte DataSource und die damit verbunden JDBC-Verbindung zu testen.

Außerdem sind im Paket `thema1.tutorial.jdbc.servlet` bereits verschiedene Code-Beispiele zur Anwendung von JDBC enthalten. Dabei ist jeweils eine Umsetzung mit einem nativen SQL-Query und eine mit einem PreparedQuery (für SQL-Queries mit Parametern) implementiert. Genaueres dazu im Anhang am Ende dieses Tutorials.

Explanation	Screenshot
18. Mit der rechten Maustaste auf das Paket <code>thema1.tutorial.jdbc.servlet</code> klicken und im Kontextmenü unter <i>New > Servlet</i> ein neues Servlet erstellen	
19. Im sich öffnenden Fenster den Klassennamen <code>ConnectionTest</code> eingeben und mit <i>Finish</i> beenden. Das neu erstellte Servlet wird geöffnet. Es soll einen Konstruktor sowie die Methoden <code>doGet()</code> und <code>doPost()</code> enthalten.	
20. Im Code zunächst die nebenstehenden Zeilen nach dem Attribut <code>serialVersionUID</code> einfügen.	<pre data-bbox="616 1623 1219 1686">@Resource(lookup="java:jboss/datasources/Shop") private DataSource dataSource;</pre> <p>Hintergrund: Diese DataSource haben wir zuvor in unserem WildFly-Server als Data Source hinterlegt. Durch die Annotation <code>@Resource</code> und das Attribut <code>lookup</code>, sorgt der Web-Container beim Aufruf der <code>DataSource</code> dafür, diese zu injizieren und nutzt dabei den JNDI (Java Naming und Directory Interface). Dabei ist die <code>DataSource</code> eine Verbindung zur hinterlegten Datenbank, wobei es möglich ist, mehrere Connections gleichzeitig aufzubauen.</p>

21. Sollte **DataSource** wegen fehlender Imports rot unterstrichen sein, die Tastenkombination **Strg+Shift+O** drücken, um alle Imports neu zu organisieren. Falls es unterschiedliche Import-Optionen gibt, die entsprechende angegebene auswählen.
 Hier:
javax.sql.DataSource. Den Dialog mit **Finish** schließen.

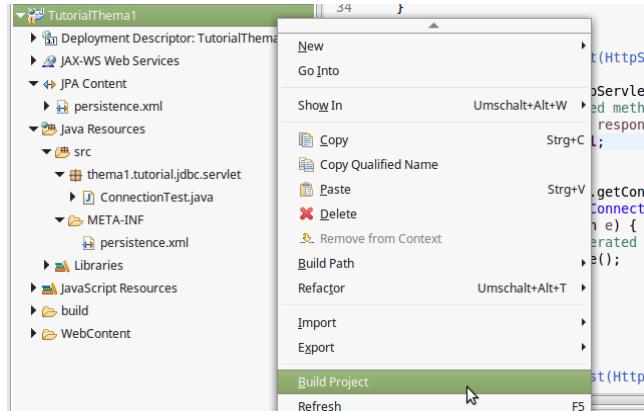


22. Nun wird die **doGet**-Methode implementiert. Dafür nebenstehenden Code in den Methoden-Rumpf kopieren.
 Anschließend fehlende Imports hinzufügen (**Strg+Shift+O**):
java.sql.Connection
 Datei speichern!

```
PrintWriter writer = response.getWriter();
Connection con = null;

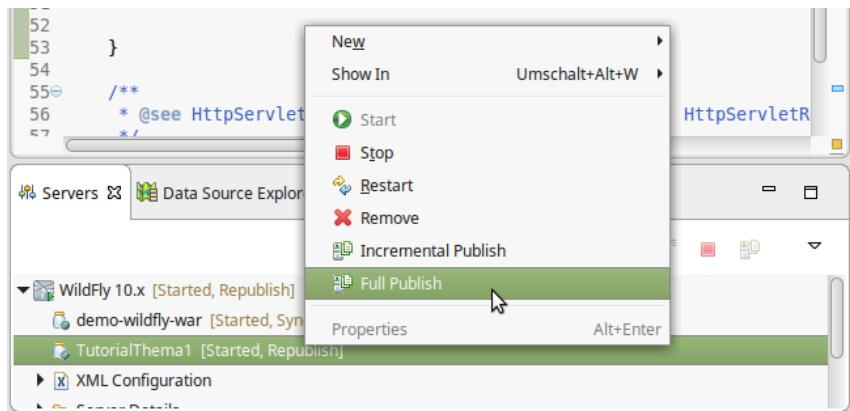
try {
    // get a Connection
    con = dataSource.getConnection();
    // print if connection is valid
    writer.println("Connection erfolgreich: " + con.isValid(10));
    // close writer and connection
    writer.close();
    con.close();
} catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

23. Das Servlet ist nun fertig und kann gestartet werden. Dafür zunächst das Projekt neu bauen:
Rechtsklick auf das Projekt > Build Project

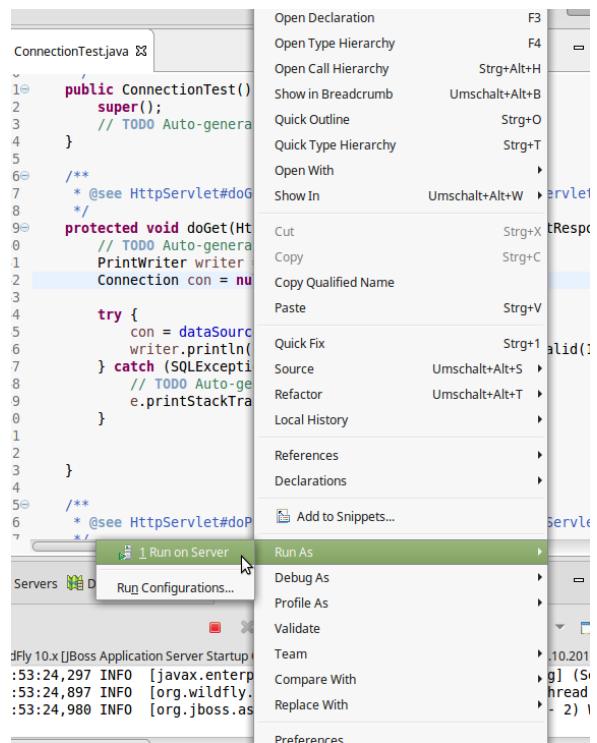


24. Nun das Projekt auf dem WildFly-Server neu publizieren:
Rechtsklick auf das Projekt im Server > Full Publish

Danach muss neben **TutorialTheme1 [Started, Synchronized]** stehen.

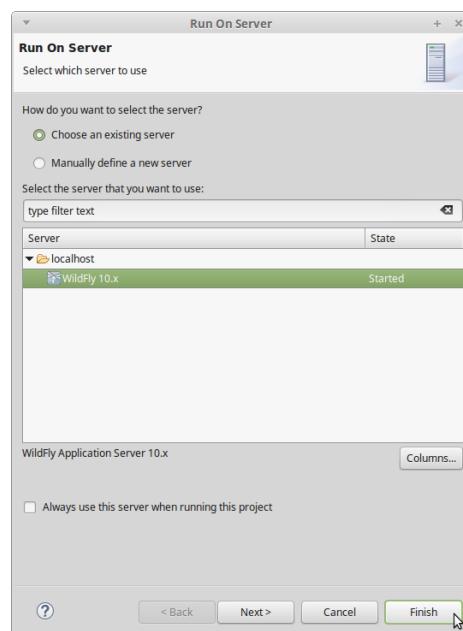


25. Nun kann das Servlet ausgeführt werden. Dafür: *Rechtsklick im Fenster des Servlet-Codes > Run as > Run on Server*



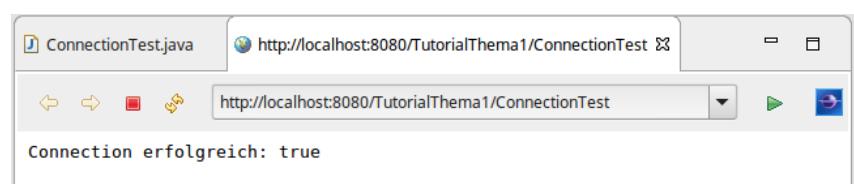
26. Im sich öffnenden Fenster den Server (Wildfly 10.x) auswählen (müsste nur einer sein, der vorausgewählt ist), und mit *Finish* bestätigen.

Am besten auch den Haken bei *Always use this server when running this project* setzen, dass nicht jedes Mal nachgefragt wird.



27. Die Ausgabe des Servers muss dem Bild rechts entsprechen.

Die DataSource wurde fehlerfrei angelegt!

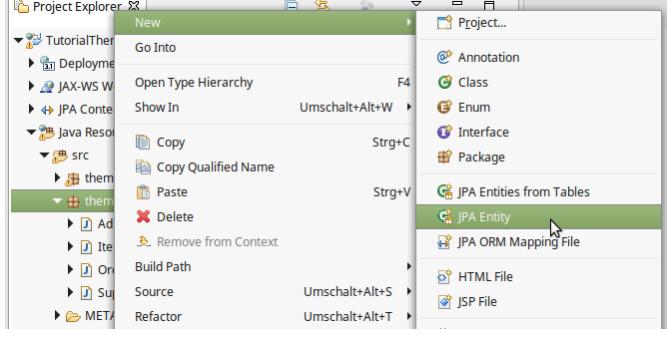
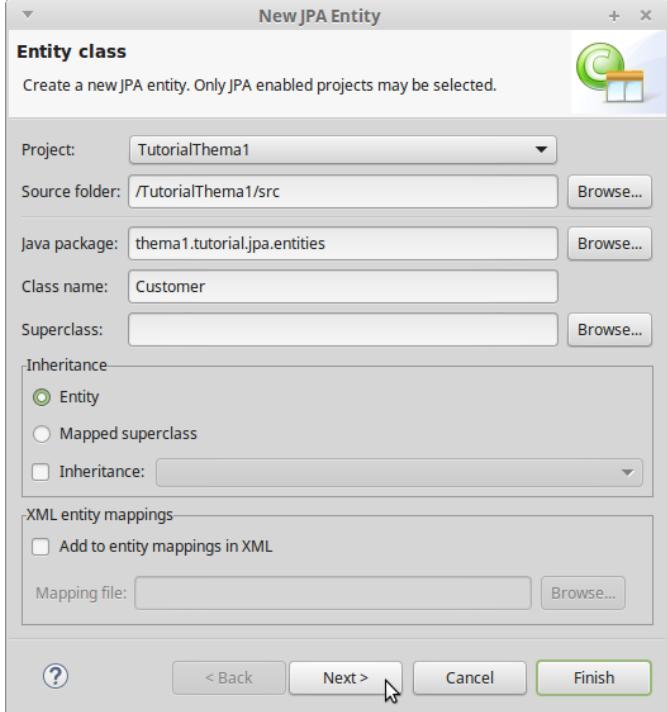


Troubleshooting:

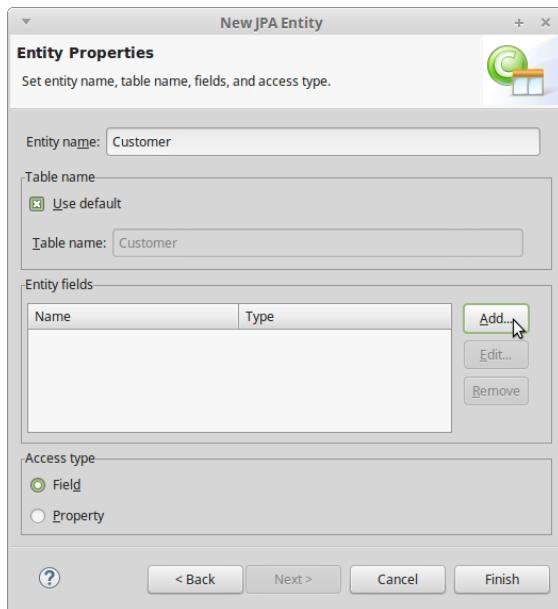
Falls eine Fehlermeldung kommt, die sagt, dass die DataSource nicht gefunden werden konnte, den Server in Eclipse noch einmal neu starten. Und vor allem überprüfen, dass der JNDI-Name der DataSource im Servlet dem entspricht, der beim Server anlegen eingegeben wurde!

TEIL 11: JPA ENTITY ERSTELLEN

In diesem Abschnitt wird das Entity „Customer“ erstellt, welche die Attribute *id*, *firstname*, *lastname*, *email*, *adresse* und *orders* hat (genaueres siehe Datenbankdiagramm in der Präsentation).

Explanation	Screenshot
<p>1. Eine neue Entity-Klasse erstellen: Rechtsklick auf das Package <code>thema1.tutorial.jpa.entities</code> > new > JPA Entity</p>	
<p>2. Im neuen Fenster den Klassennamen <code>Customer</code> eintragen und mit <i>Next</i> weiter navigieren.</p>	

3. Nun können die Attribute des Entitys angelegt werden. Dazu den Button *Add...* klicken.

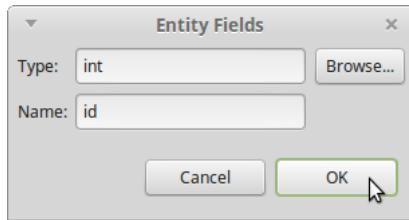


4. Im kleinen Fenster die Attribute des Fields setzen:

Type: int

Name: id

Mit *OK* bestätigen.



5. Die Schritte 3 und 4 für alle weiteren Attribute wiederholen entsprechend der nebenstehenden Tabelle.

Type	Name
String	firstname
String	lastname
String	email
Adresse	adresse
List<Orders>	orders

6. Die Liste sollte dann der nebenstehenden Liste entsprechen.

Abschließend mit *Finish* beenden.

Hinweis:

Warnungen bezüglich dem Datentyp *Adresse* können ignoriert werden.

Hintergrund:

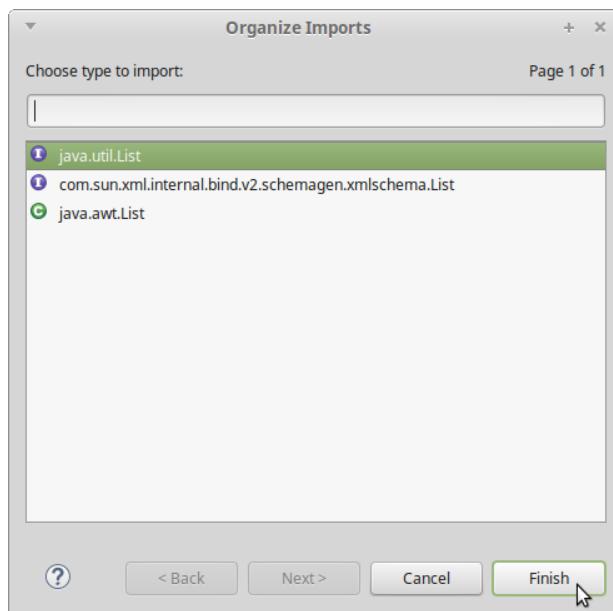
Es wird eine neue Entity-Klasse erstellt, welche bereits alle angelegten Attribute, einen leeren Konstruktor, sowie Getter und Setter für alle Attribute beinhaltet.



7. Es müssen die Imports neu organisiert werden (*Strg + Shift + O*).

In der Liste `java.util.list` auswählen.

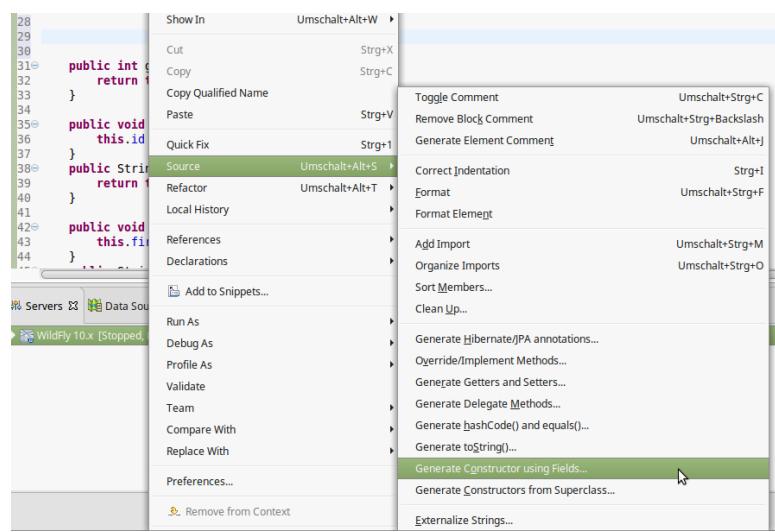
Mit *Finish* beenden. Jetzt dürfte es keine Fehler mehr geben.



8. Die Funktion `setId()` muss gelöscht werden, da wir später die id automatisch generieren lassen wollen.

9. Nun soll ein Konstruktor hinzugefügt werden, bei dem Attribute übergeben werden können. Dazu nutzen wir die Code-Generierung von Eclipse.

Dazu unterhalb des leeren Konstruktors:
Rechtsklick > Source > Generate Constructor using Fields...

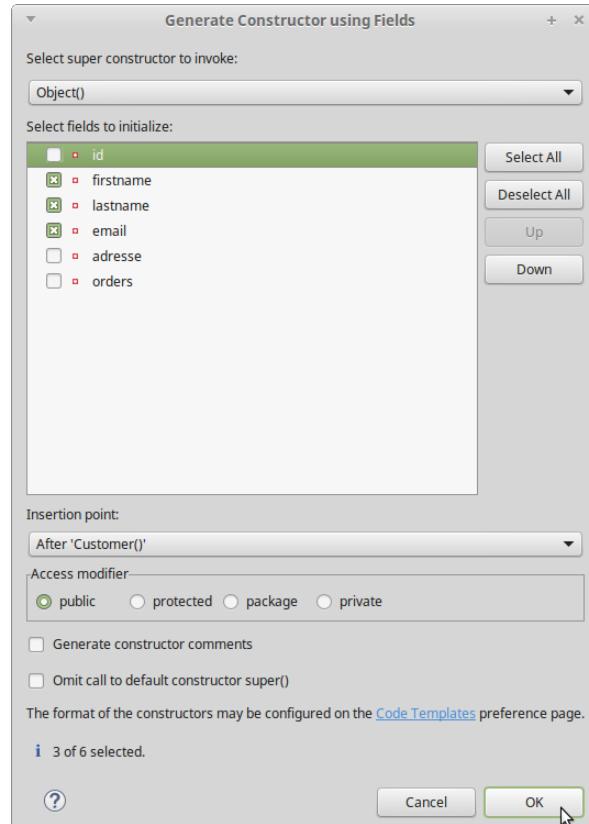


10. Aus der Liste der Attribute **alle außer id, adresse und orders** auswählen und mit *OK* bestätigen.

Rechts unten abgebildeter Code sollte nun in der Entity-Klasse hinzugefügt worden sein.

Hintergrund:

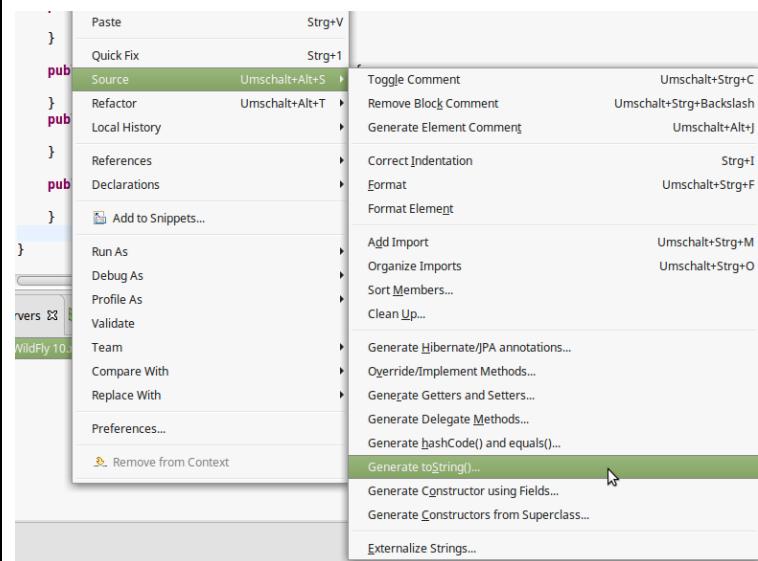
Das Feld `id` wird nicht mit ausgewählt, da dieses beim persistieren automatisch generiert wird.
 Die Felder `adresse` und `orders` werden nach dem Erzeugen über die Setter-Methoden gesetzt.



```
public Customer(String firstname, String lastname, String email) {
    super();
    this.firstname = firstname;
    this.lastname = lastname;
    this.email = email;
}
```

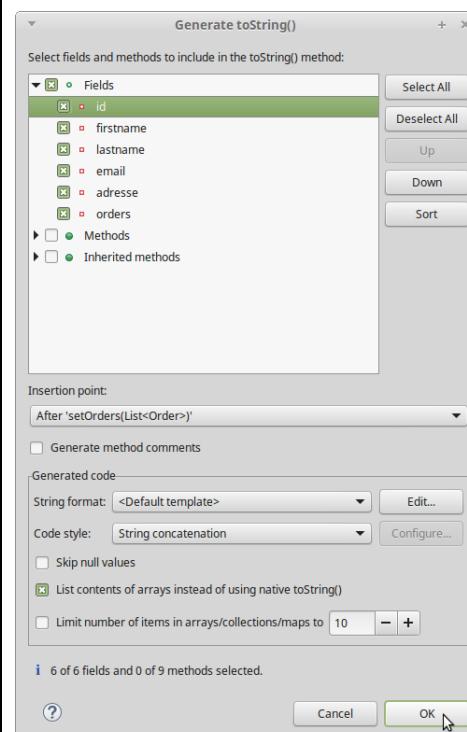
11. Nun soll für die einfachere Ausgabe der Entity eine `ToString()`-Methode implementiert werden. Auch dafür nutzen wir die Code-Generierung von Eclipse.

Dafür *Rechtsklick > Source > Generate toString()...*



12. Im sich öffnenden Fenster prüfen, dass die Auswahl der auf dem Screenshot entspricht und anschließend auf *OK* klicken.

Der generierte Code muss dem rechts unten abgebildeten Code entsprechen.

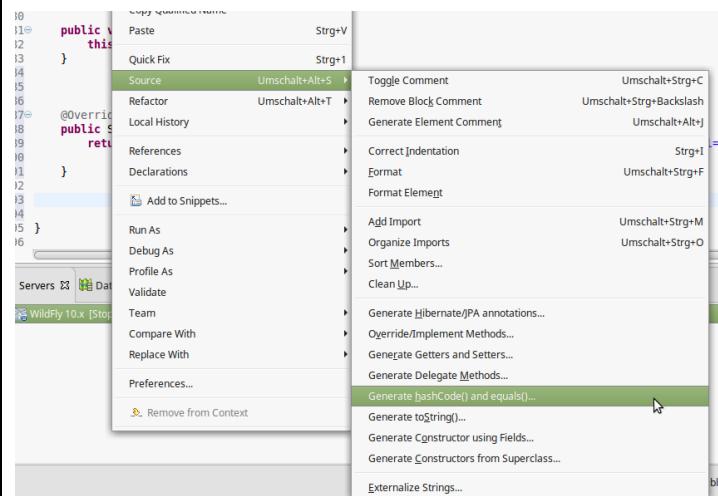


```

@Override
public String toString() {
    return "Customer [id=" + id + ", firstname=" + firstname + ", lastname=" + lastname + ", email=" + email
           + ", adresse=" + adresse + ", orders=" + orders + "]";
}
  
```

13. Für den Objekt-Vergleich sind nun noch die Methoden `hashCode()` und `equals()` notwendig, welche mit der Code-Generierung erzeugt werden.

Dafür *Rechtsklick > Source > Generate hashCode() and equals()...*

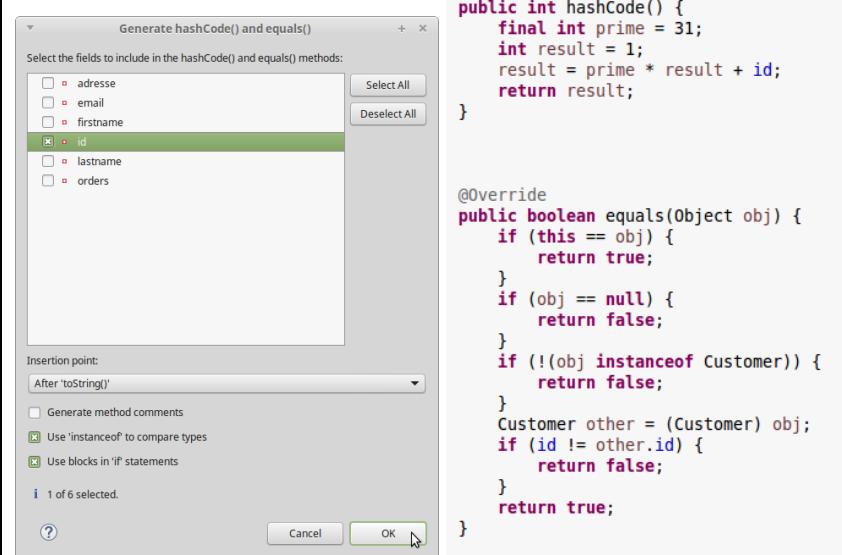


14. Für den Objektvergleich wollen wir nur das Attribut `id` nutzen.

Dafür *DeselectAll* klicken und dann `id` auswählen.

Mit *OK* bestätigen.

Der generierte Code muss nebenstehendem entsprechen.



Nun sind alle Attribute und Methoden definiert. Es fehlen noch die Annotationen an den Attributen, welche in den folgenden Schritten hinzugefügt werden.

15. Das Attribut `id` soll der Primärschlüssel sein (durch `@id`) und automatisch generiert werden (durch `@GeneratedValue(...)`).

Dafür oberhalb des Attributs `id` die ersten beiden Zeilen des nebenstehenden Codes einfügen und anschließend die Imports anpassen (*Strg+Shift+O*).

```

@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private int id;

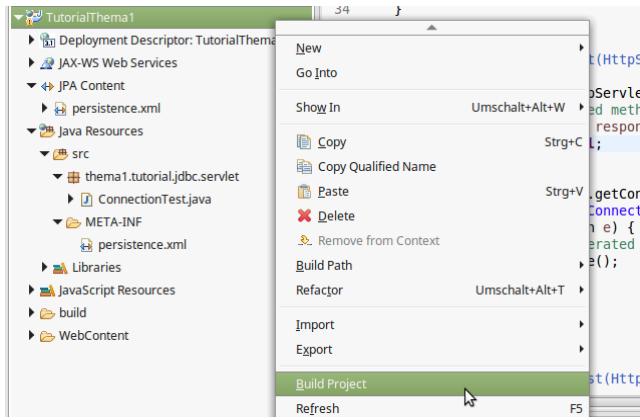
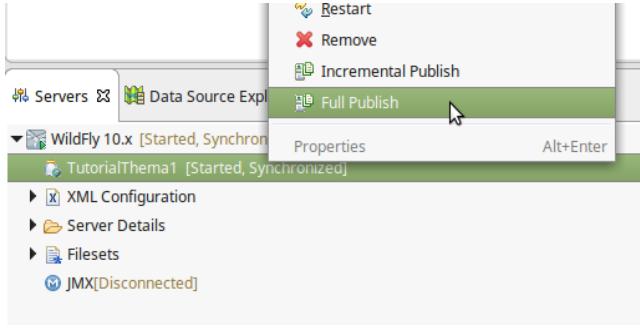
```

16. Die Attribute `firstname`, `lastname` und `email` sollen keine Nullwerte erlauben, was durch die Annotation `@Column` definiert wird.

```

@Column(nullable=false)
private String firstname;
@Column(nullable=false)
private String lastname;
@Column(nullable=false)
private String email;

```

<p>Dazu vor die drei Attribute entsprechend des nebenstehenden Codes die Annotation hinzufügen.</p>	
<p>17. Die Entity-Beziehungen werden über Annotation hinzugefügt. Dafür vor dem Attribut <code>adresse</code> die Annotation <code>@OneToOne</code> und vor das Attribut <code>orders</code> die Annotation <code>@OneToMany</code> hinzufügen (entsprechend nebenstehendem Code).</p>	<pre><code>@OneToOne(cascade = CascadeType.PERSIST) private Adresse adresse; @OneToMany(cascade = CascadeType.PERSIST, fetch=FetchType.EAGER) private List<Orders> orders;</code></pre> <p>Hintergrund: Über den CascadeType wird festgelegt, dass die Liste mit Orders automatisch persistiert wird, wenn der Customer persistiert wird. Über den FetchType wird festgelegt, dass die Liste mit Orders immer mit selektiert werden soll, wenn der Customer selektiert wird.</p>
<p>In der Datei <code>persistence.xml</code> ist über ein Attribut festgelegt, dass die Entitäten beim Start der Applikation in der Datenbank automatisch erstellt werden und beim Beenden gelöscht werden. Im Folgenden prüfen wir, ob diese Eigenschaft erfüllt wird, indem wir zunächst den aktuellen Stand auf den Server publizieren und anschließend in der Datenbank die Existenz der Tabellen nachvollziehen.</p>	
<p>18. Das Projekt neu bauen: Datei speichern, dann <i>Rechtsklick auf das Projekt > Build Project</i></p>	
<p>19. Den neuen Stand des Projekts auf den Server publizieren: <i>Rechtsklick auf das Projekt im Server-Fenster > Full Publish</i></p> <p>Hinweis: Dabei werden in der Konsolenausgabe verschiedene Fehler angezeigt. Sofern diese ähnlich zu nebenstehendem Beispiel sind, können sie ignoriert werden. Diese Fehler entstehen dadurch, dass durch das Attribut <code>create-drop</code> (in der Datei <code>persistence.xml</code> gesetzt) versucht wird, beim Starten alle Tabellen zu löschen, wobei in der Datenbank keinerlei Tabellen enthalten sind.</p>	 <p>09:19:58,886 ERROR [org.hibernate.tool.hbm2ddl.SchemaExport] (ServerService Thread Pool -- 60) You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'Order (id)' at line 1</p>

20. Im Terminal, indem bereits vorher die Tabellen der Datenbank geprüft wurden, alle Tabellen anzeigen lassen:

In der Datenbank in das Schema shop wechseln:
use shop;

Dann alle Tabellen anzeigen lassen:
show tables;

Hintergrund:

Es wurden allen Entities, die im Eclipse-Projekt existieren, automatisch als Tabellen angelegt. Außerdem wurde für alle Attribute, die eine @ManyToOne oder @ManyToMany Annotation haben, eine Join-Tabelle erstellt, wie Customer_Order und Order_Item.

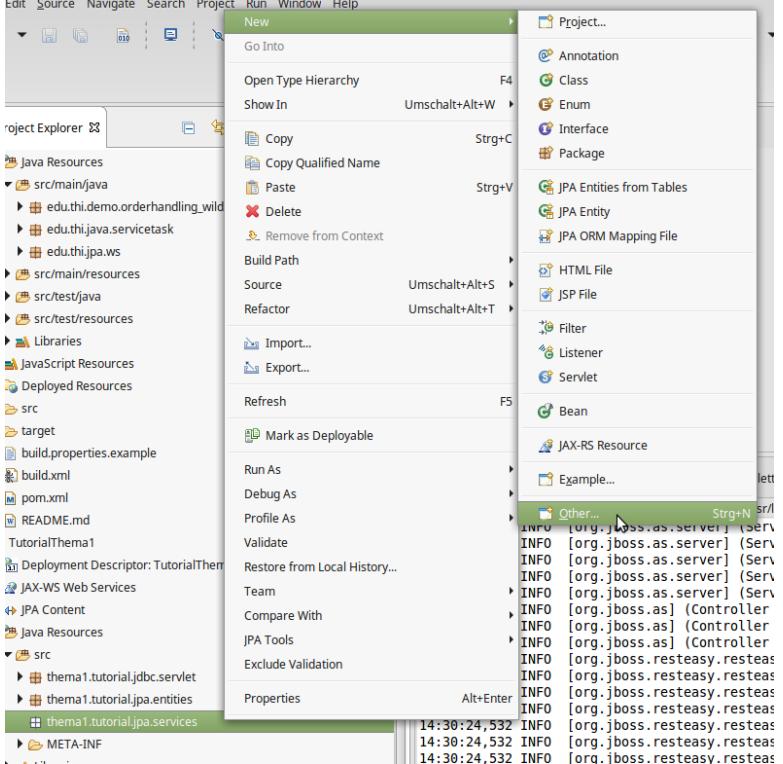
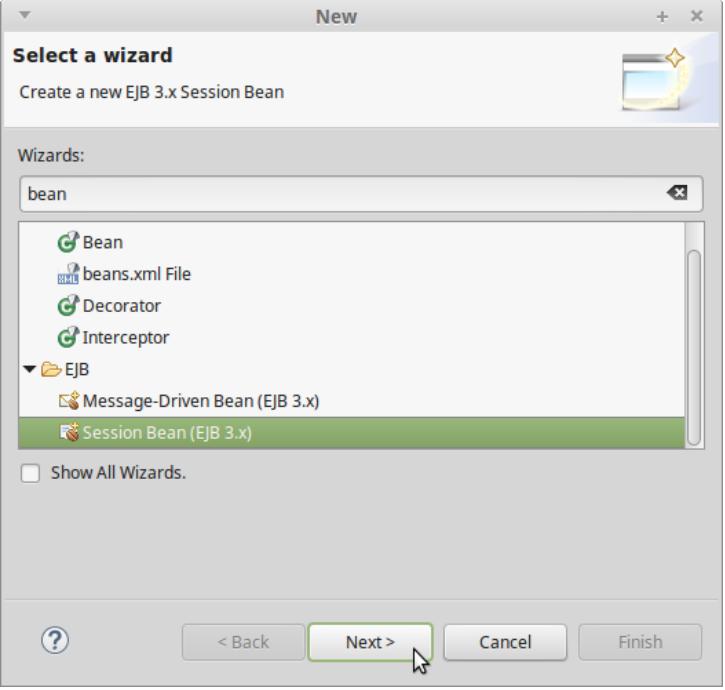
```
MariaDB [(none)]> use shop;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [shop]> show tables;
+-----+
| Tables_in_shop |
+-----+
| Adresse
| Customer
| Customer.Orders
| Item
| Orders
| Orders.Item
| Supplier
| hibernate_sequence |
+-----+
8 rows in set (0.00 sec)

MariaDB [shop]> █
```

TEIL 12: SERVICE KLASSE ZUR ENTITY CUSTOMER ERSTELLEN

Im folgenden Abschnitt wird ein Service erstellt, der alle notwendigen Funktionen implementiert, um die Entität Customer zu nutzen. Dafür wird ein Session Bean erstellt, wobei eine Serviceklasse und ein entsprechendes Interface (mit Endung ...Local) erzeugt wird. Diese Klassen werden später vom Servlet aus genutzt, um Entities zu erstellen, zu bearbeiten oder zu löschen.

Explanation	Screenshot
<p>1. Erstellen einer neuen Service-Klasse:</p> <p><i>Rechtsklick auf das Paket thema1.tutorial.jpa.services > New > Other</i></p>	
<p>2. Im Dialog nach bean suchen und im Ordner EJB „Session Bean“ auswählen.</p> <p>Weiter mit <i>Next</i>.</p>	

3. Als **Klassennamen** CustomerService eintragen.

Bei **Local** ein Kreuz setzen.

Mit *Finish* beenden.

Hintergrund:

Durch das Kreuzchen bei Local wird automatisch ein Interface für die Service-Klasse implementiert, die wir später im Servlet nutzen werden.



4. In der Datei CustomerService wollen wir nun eine Methode zum Erstellen eines neuen Kunden implementieren.

Dazu muss zunächst der EntityManager injiziert werden.

Dafür die zwei unteren Zeilen des nebenstehenden Codes unterhalb des Klassenkopfes einfügen und die Imports erneuern (*Strg+Shift+O*)

```
public class CustomerService implements CustomerServiceLocal {  
  
    @PersistenceContext  
    EntityManager em;
```

5. Unterhalb des Konstruktors sollen nun zwei Klassen implementiert werden.

Die erste Klasse (**insert**) soll den übergebenen Customer speichern (mit **persist**).

Damit wir überprüfen können, ob unser neu angelegter Kunde wirklich persistiert wurde, schreiben wir die zweite Funktion (**selectAll**), welche alle Kunden aus der Tabelle selektiert.

```
@Override  
public Customer insert(Customer c){  
    em.persist(c);  
    return c;  
}  
  
@Override  
public List<Customer> selectAll(){  
    TypedQuery<Customer> query = em.createQuery("SELECT c FROM Customer c", Customer.class);  
    List<Customer> result = query.getResultList();  
    return result;  
}
```

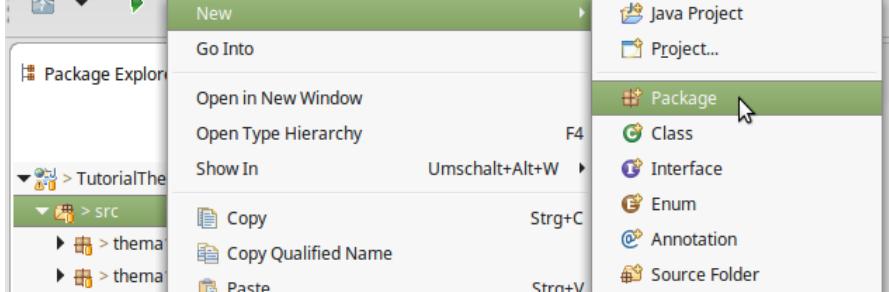
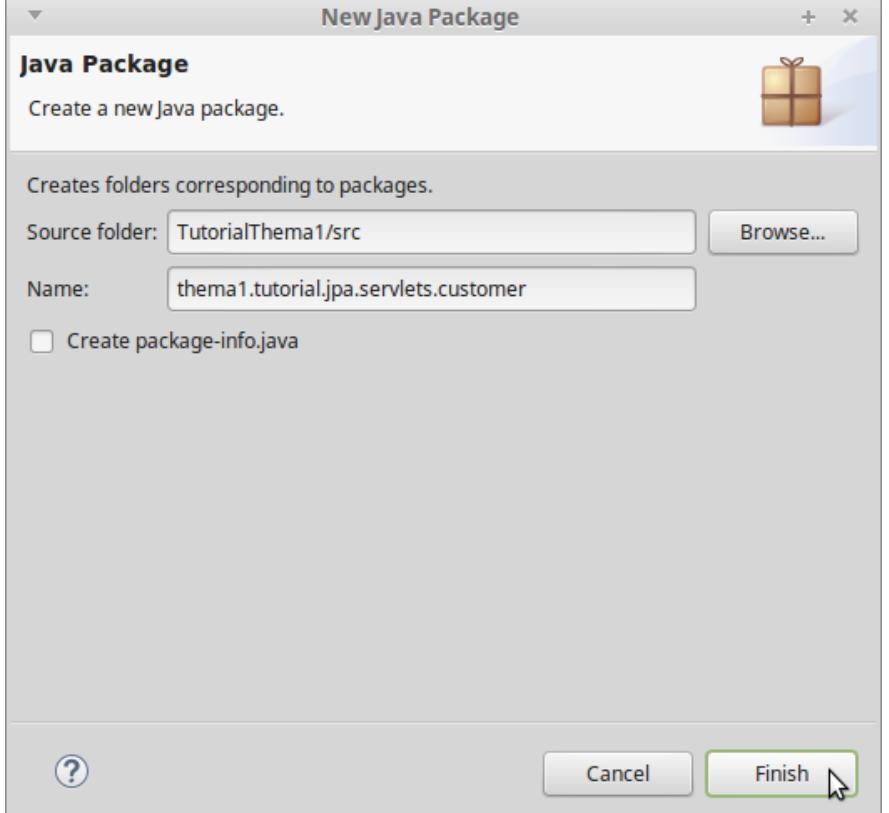
Hintergrund:

In der Funktion **selectAll()** nutzen wir die Abfragesprache JPQL, da es bei JPA keine Methode gibt, die alle Einträge einer Tabelle selektiert. Dafür wird eine getypte Abfrage erstellt (alle Einträge sollen als Customer-Entity gelesen werden). Durch **getResultList()** werden alle Ergebnisse der Abfrage als Liste

<p>Nebenstehenden Code unterhalb des Konstruktors hinzufügen. Und abschließend Imports reorganisieren (<i>Strg+Shift+O</i>). Für die Liste wieder <code>java.util.list</code> wählen.</p>	<p>zurückgegeben. Über <code>getSingleResult()</code> kann nur das erste Element zurückgegeben werden.</p>
<p>6. Die erstellten Funktionen müssen nun auch dem Interface hinzugefügt werden. Dazu in der Datei <code>CustomerServiceLocal</code> die zwei Zeilen der Funktionsdeklarationen entsprechend nebenstehendem Code hinzufügen.</p> <p>Danach die Imports erneuern (<i>Strg+Shift+O</i>) und für <code>List</code> wieder <code>java.util.List</code> auswählen.</p>	<pre><code>@Local public interface CustomerServiceLocal { public Customer insert(Customer c); public List<Customer> selectAll(); }</code></pre>

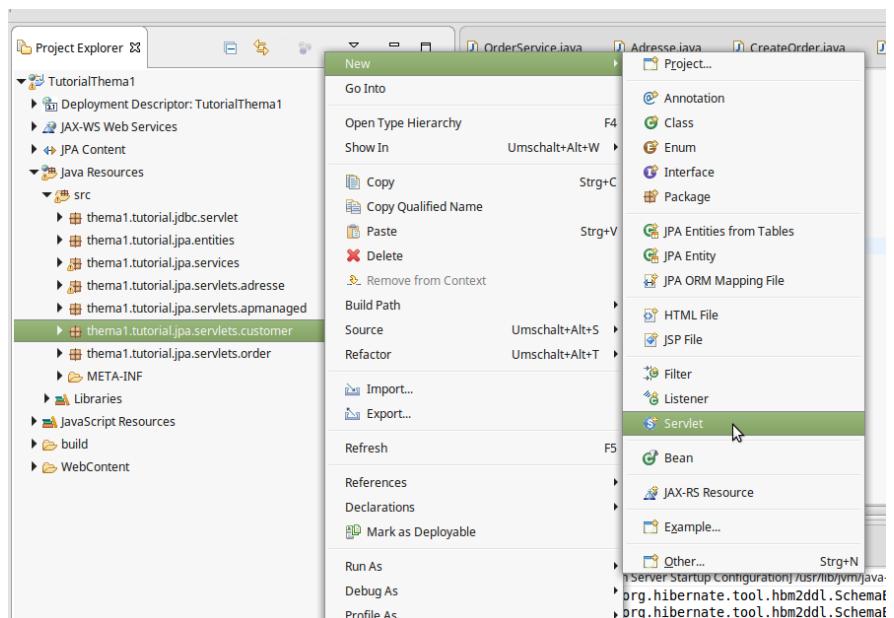
TEIL 13: SERVLET ERSTELLEN

Im folgenden Abschnitt wird ein Servlet erstellt, welches einen neuen Customer erstellt und anschließend alle Einträge der Tabelle Customer anzeigt, um zu prüfen, ob auch wirklich ein neuer Eintrag erstellt wurde.

Explanation	Screenshot
<p>1. Neues Paket für Servlets der Entity Customer erstellen.</p> <p><i>Rechtsklick auf src > New > Package</i></p>	
<p>2. Namen des Packages eingeben:</p> <p><code>tHEMA1.tUTORIAL.JPA.SERVLETS.CUSTOMER</code></p> <p>Mit <i>Finish</i> beenden</p>	

3. Erstellen eines neuen Servlets:

Rechtsklick auf das Paket `tHEMA1.tUTORIAL.JPA.SERVLETS.CUSTOMER` > New > Other



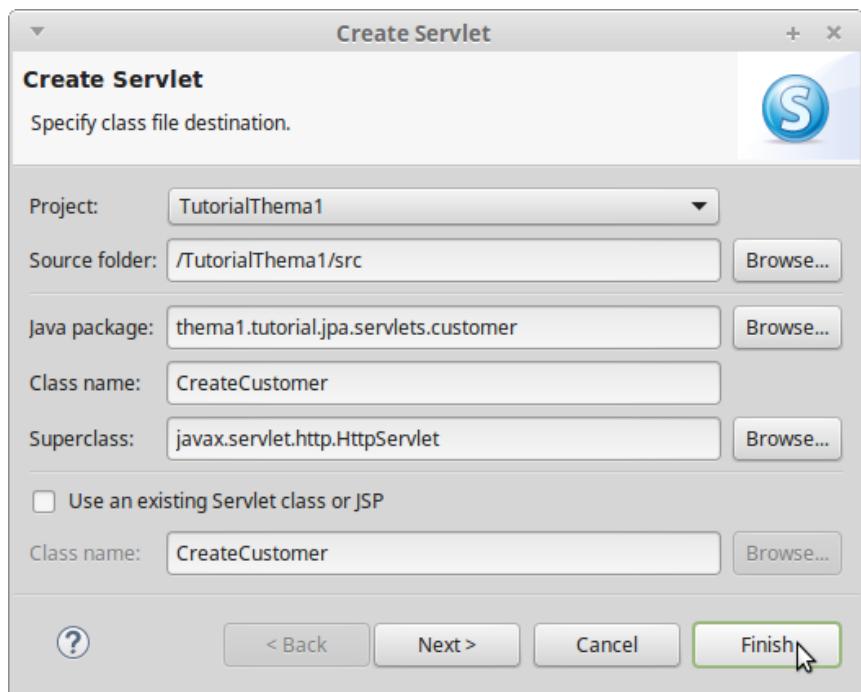
4. Den Klassennamen eintragen: `CreateCustomer`

Weiter mit *Finish*.

Das erstellte Servlet beinhaltet bereits einen Konstruktor, eine `doGet()` und eine `doPost()` Methode.

Hintergrund:

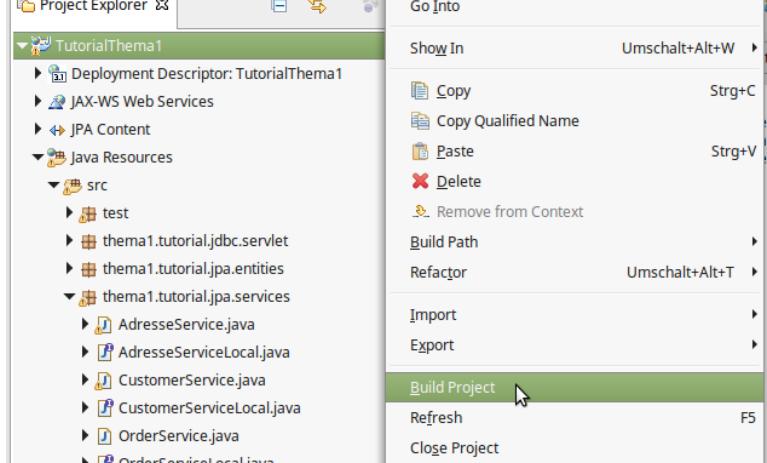
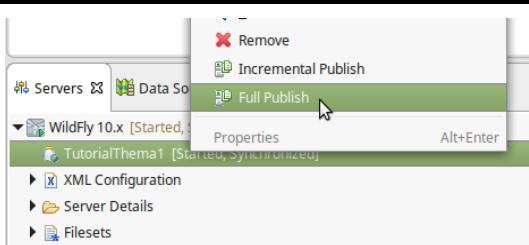
Die `doGet()`-Methode wird aufgerufen, wenn eine Anfrage über GET erfolgt, z.B. Aufruf über URL. Die `doPost()`-Methode wird ausgeführt, wenn eine POST-Anfrage erfolgt, z.B. über das Absenden eines Formulars.



5. Es wird eine neue Servlet-Klasse erzeugt, die bereits einen Konstruktor und die Methoden `doGet()` und `doPost()` beinhaltet.

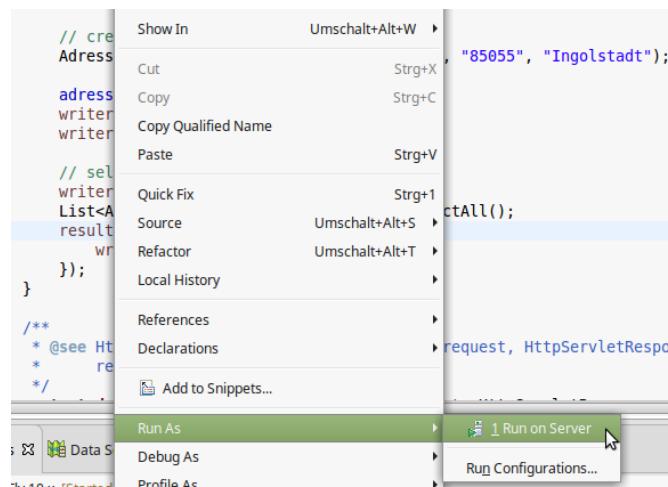
Wir erzeugen eine neue Klassenvariable vom Typ des Interfaces des `AdresseServices`. Dazu die nebenstehenden Code-Zeilen zwischen die Variable `serialVersionUID` und den Konstruktor einfügen. Und anschließend die Imports aktualisieren.

```
@Inject
CustomerServiceLocal customerService;
```

<p>Somit können wir alle in dem Interface angebotenen Methoden nutzen und müssen im Servlet keine JPA-Methoden anwenden.</p>	
<p>6. Nebenstehenden Code in die <code>doGet()</code>-Methode kopieren und die Imports neu organisieren. Für die Aktualisierung der Imports folgendes importieren: <code>java.sql.date</code> <code>java.util.list</code> Datei speichern!</p> <p>Hintergrund: Zunächst wird ein neues Objekt <code>Customer</code> erstellt. Diesem wird eine Liste mit Bestellungen hinzugefügt sowie eine Adresse. Anschließend wird das Objekt persistiert. Am Ende werden alle Ergebnisse der Tabelle <code>Customer</code> gelesen, um zu prüfen, ob der Eintrag hinzugefügt wurde.</p>	<pre>final PrintWriter writer = response.getWriter(); // create new Customers Customer c1 = new Customer("Max", "Mustermann", "max@mustermann.de"); Orders o = new Orders(new Date(Calendar.getInstance().getTime().getTime())); List<Orders> olist = new ArrayList<>(); olist.add(o); c1.setOrders(olist); Adresse a = new Adresse("Hauptstrasse", "3", "85049", "Ingolstadt"); c1.setAdresse(a); customerService.insert(c1); writer.println("Created Customers:"); writer.println(c1.toString()); // select all Customers from table writer.println("Current Table Content:"); List<Customer> result = customerService.selectAll(); result.forEach((customer) -> { writer.println(customer.toString()); });</pre>
<p>7. Nun ist das Servlet fertig und kann ausgeführt werden. Dazu muss zunächst das Projekt neu gebaut werden.</p> <p><i>Rechtsklick auf den Projektordner > Build Project</i></p>	
<p>8. Nun das Projekt neu auf den Server publizieren:</p> <p><i>Rechtsklick auf das Projekt im WildFly-Server > Full Publish</i></p>	

9. Nun kann das Servlet gestartet werden:

*Rechtsklick in der Servlet-Datei
> Run as > Run on Server*



10. Die Ausgabe sollte der nebenstehenden entsprechen.

```
Created Customers:
Customer [id=1, firstname=Max, lastname=Mustermann,
email=max@mustermann.de, adresse=Adresse [id=2, street=Hauptstrasse,
houseNumber=3, postalCode=85049, city=Ingolstadt], orders=[Order
[id=3, generated=2018-11-08, items=null, supplier=null]]]
Current Table Content:
Customer [id=1, firstname=Max, lastname=Mustermann,
email=max@mustermann.de, adresse=Adresse [id=2, street=Hauptstrasse,
houseNumber=3, postalCode=85049, city=Ingolstadt], orders=[Order
[id=3, generated=2018-11-08, items=[], supplier=null]]]
```

11. Das Ergebnis kann auch in der Datenbank direkt überprüft werden:

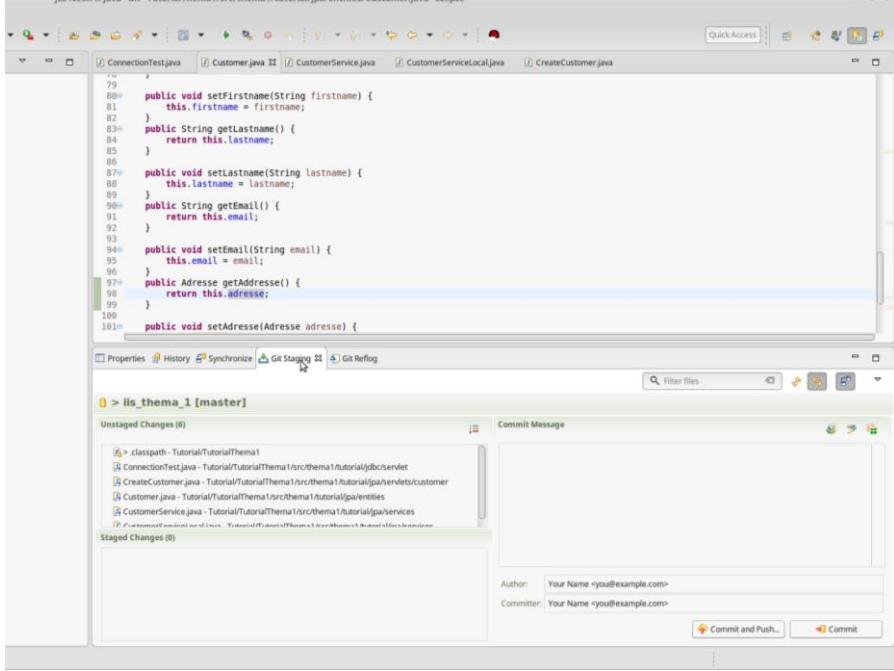
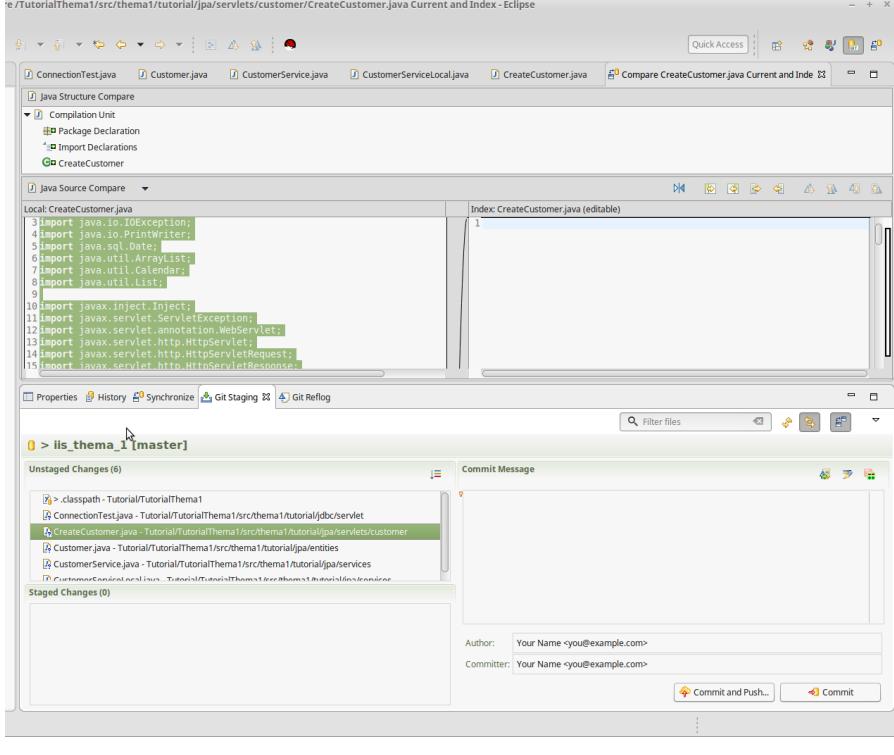
```
select * from Customer;
select * from Adresse;
select * from Orders;
```

```
MariaDB [shop]> select * from Customer;
+----+-----+-----+-----+
| id | email           | firstname | lastname  | adresse_id |
+----+-----+-----+-----+
| 1  | max@mustermann.de | Max       | Mustermann |          2 |
+----+-----+-----+-----+
1 row in set (0.00 sec)
```

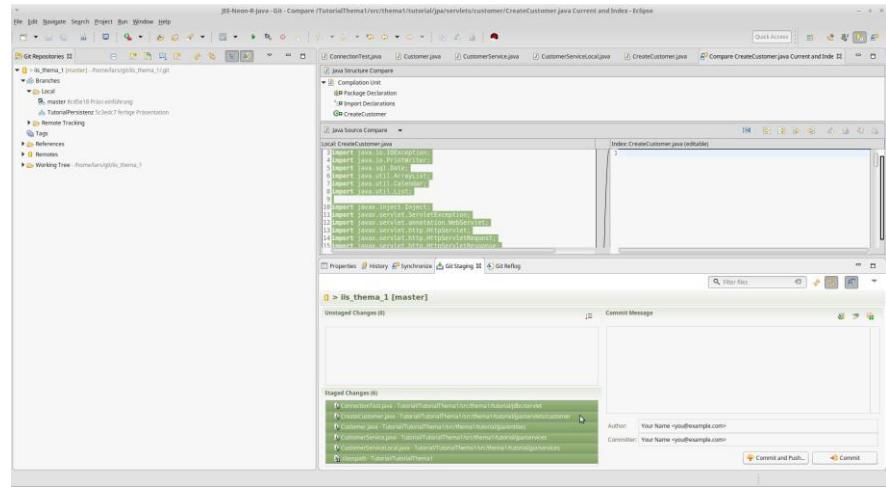
```
MariaDB [shop]> select * from Adresse;
+----+-----+-----+-----+
| id | city      | houseNumber | postalCode | street      |
+----+-----+-----+-----+
| 2  | Ingolstadt | 3           | 85049      | Hauptstrasse |
+----+-----+-----+-----+
1 row in set (0.00 sec)
```

```
MariaDB [shop]> select * from Orders;
+----+-----+-----+
| id | generated | supplier_id |
+----+-----+-----+
| 3  | 2018-11-08 |          NULL |
+----+-----+-----+
1 row in set (0.00 sec)
```

TEIL 14: DEN FERTIGEN CODE COMMITEN UND PUSHEN

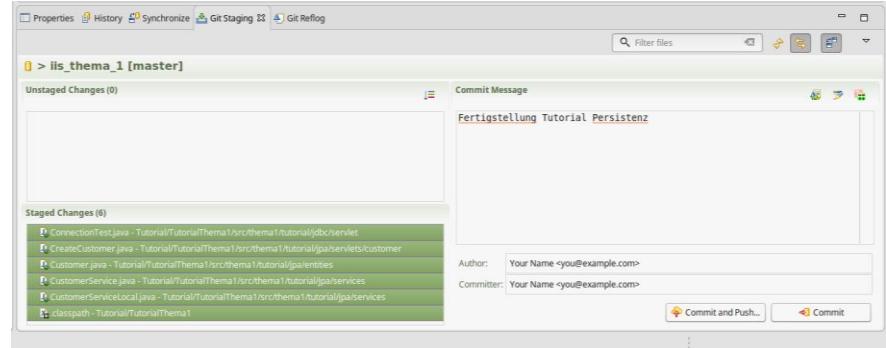
Explanation	Screenshot
<ol style="list-style-type: none">Nach Abschluss des Tutorials wieder in die Git Ansicht wechseln und rechts unten die Lasche Git Staging auswählen.	
<ol style="list-style-type: none">Alle geänderten Dateien werden aufgelistet. Die jeweiligen Änderungen können per Doppelklick auf eine Datei angezeigt werden. <p>Hinweis: Eclipse legt die kompilierten .class Dateien in dem Ordner build ab. Zusätzlich wird die eine .gitignore Datei angelegt, die verhindert, dass die classes im Repository landen.</p> <p>Manchmal muss die .gitignore per Hand angelegt oder angepasst werden. Die Website https://www.gitignore.io/ bietet an, .gitignore Dateien für gewählte Entwicklungsumgebungen und Tools (z.B. Maven) zu generieren. Achte immer darauf, keine unnötigen Dateien in das Repository mit aufzunehmen.</p>	

3. Die geänderten Dateien markieren und in den Bereich „Staged Changes“ ziehen.

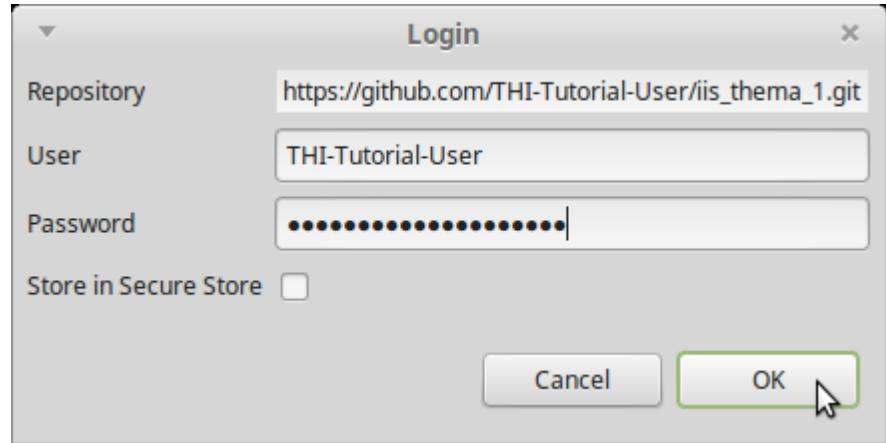


4. In der rechten Box die Commit Nachricht „Fertigstellung Tutorial Persistenz“ eingeben. Anschließend die Angaben bei Author und Committer überprüfen.

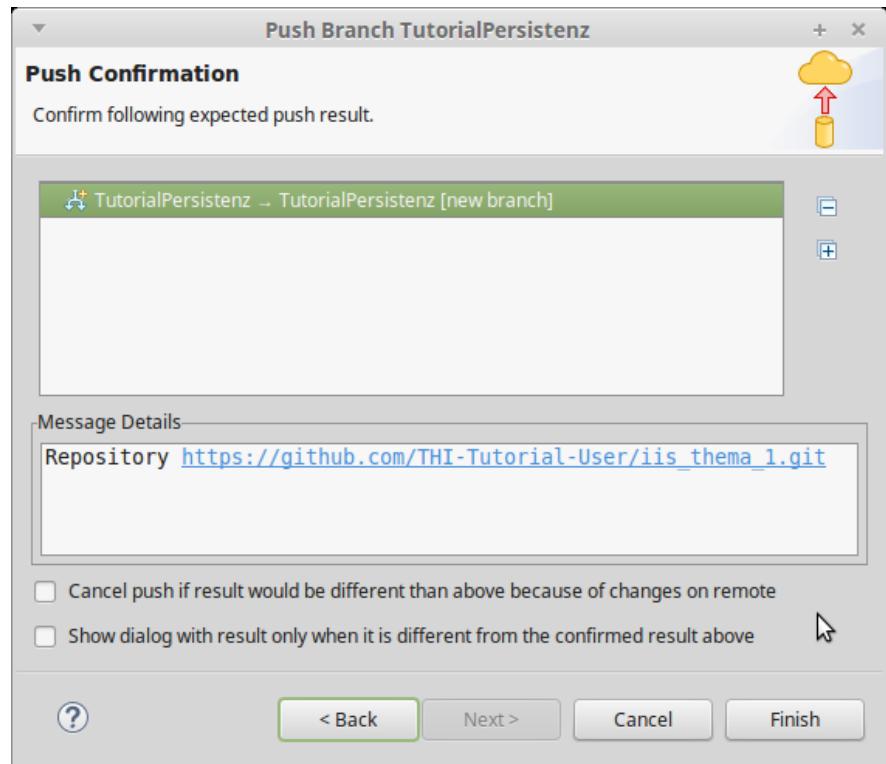
Klick auf den Button „Commit and Push“



5. In dem Dialog auf weiter. Nach Aufforderung mit dem GitHub User einloggen.



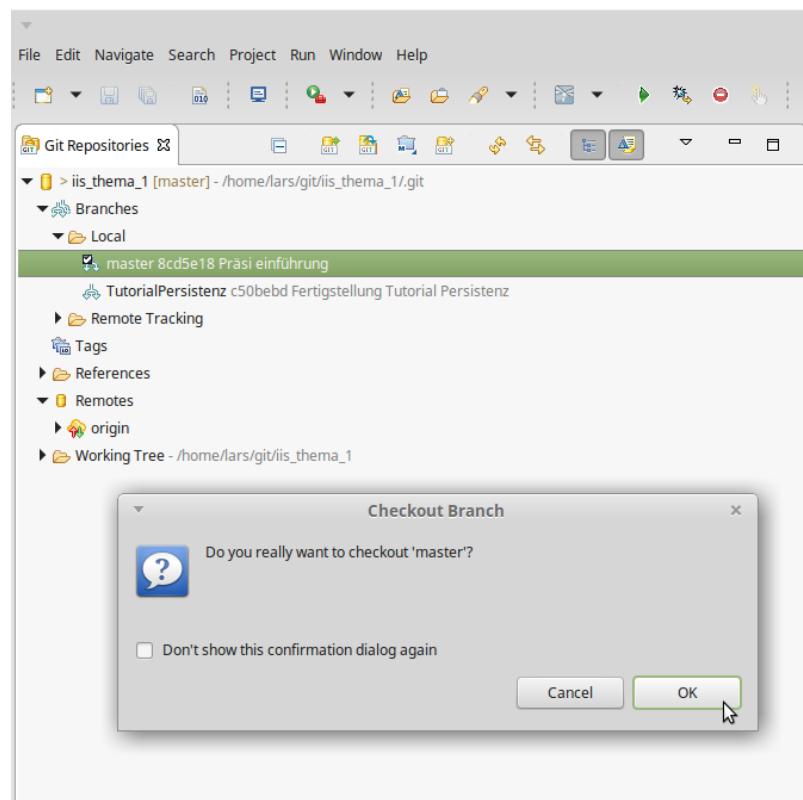
6. Im letzten Dialog nur auf „Finish“ gehen.



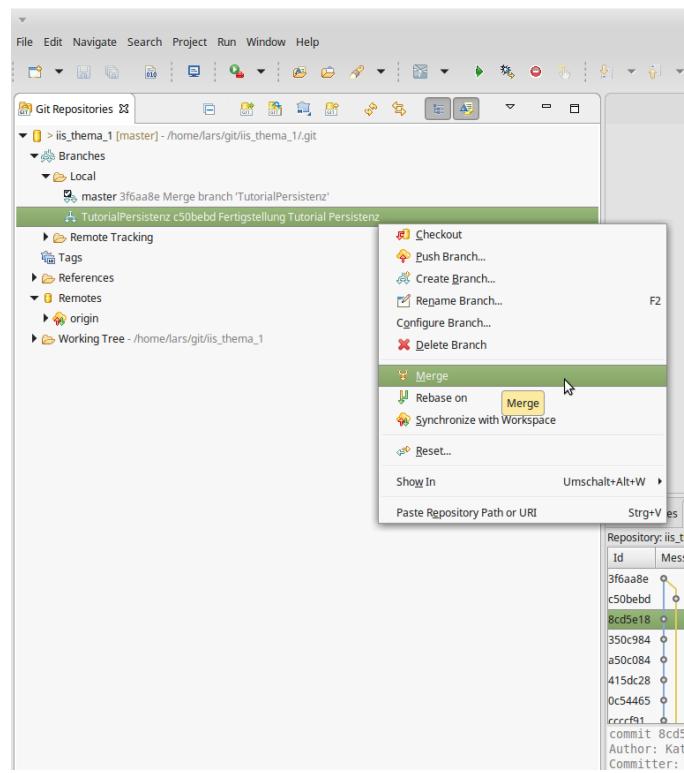
7. Die neue Version des Branches ist nun auch in dem GitHub Repository verfügbar.

So lässt sich parallel an verschiedenen Features arbeiten und den Teammitgliedern eine Möglichkeit geboten, den aktuellen Fortschritt einzusehen und mitzuhelfen.

8. Nachdem der Kontext, in dem der Branch erstellt wurde, abgeschlossen ist, kann er in den Hauptbranch, hier master, zurückgemerget werden: Zuerst wieder nach master wechseln per Doppelklick auf den Branch. Den Dialog bestätigen.

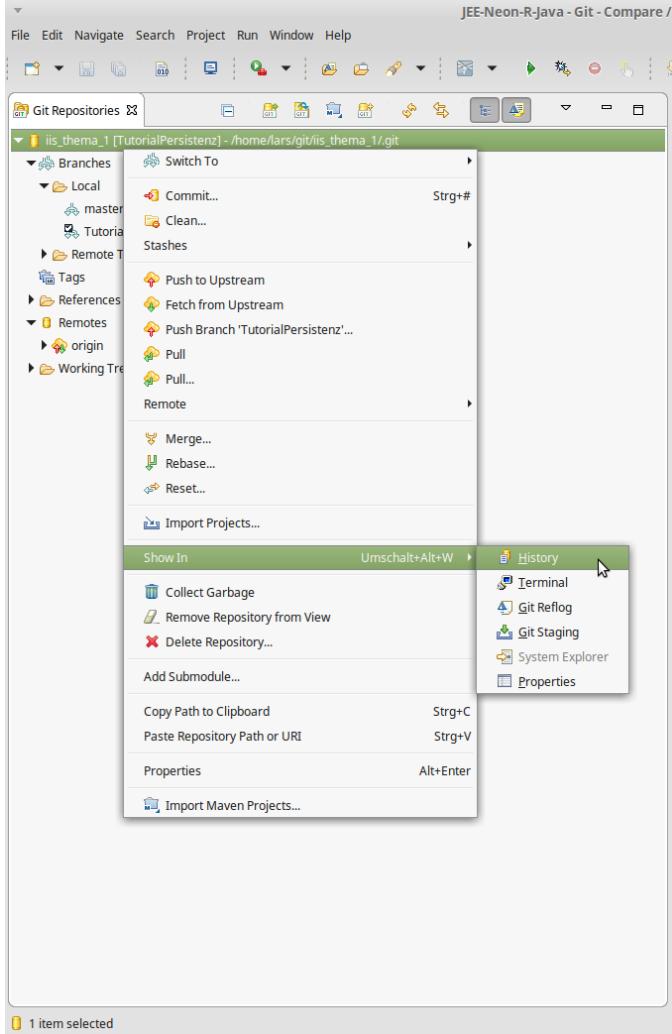
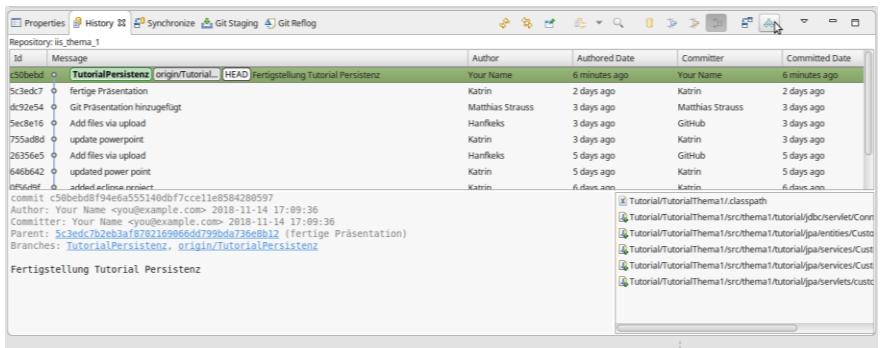


9. Rechtsklick auf den Branch der gemerget werden soll, und merge auswählen. Ein Dialog zeigt das Ergebnis des merge an. Nach dem merge kann der master branch auch gepusht werden. (Kontextmenü des Branch > Push Branch...)



TEIL 15: DIE GIT HISTORY PER EGIT

Dieser Abschnitt beschreibt, wie Git Clients dabei helfen können, den Projektverlauf nachzuvollziehen.

Explanation	Screenshot
<ol style="list-style-type: none">1. Die History des Repositorys per Kontextmenü des Repositorys in der Git View öffnen. <p>Hinweis: Es ist genauso möglich, die History einer einzelnen Datei zu betrachten. Dafür im Kontextmenü auf „Team“ > „Show in history“ gehen.</p>	
<ol style="list-style-type: none">2. Um alle Branches zu sehen das Branch Symbol auswählen. <p>Hinweis: Der Kommandozeilen Befehl für eine ähnliche Ansicht lautet:</p> <pre>git log --all --graph --decorate --oneline --simplify-by-decoration</pre>	

3. Nach der Auswahl eines Commits werden die geänderten Dateien rechts unten angezeigt.

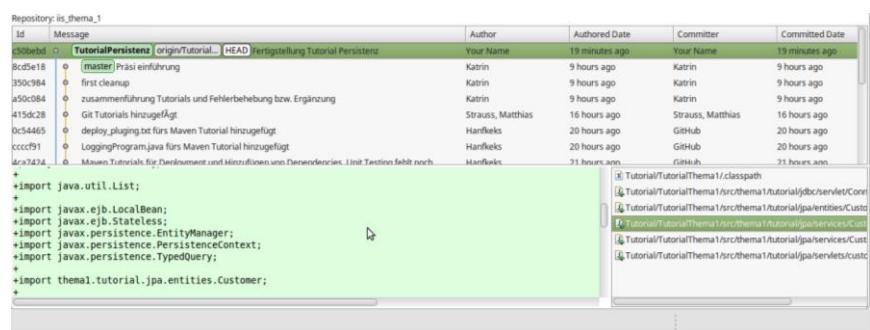
Der Diff zwischen der Version einer Datei im ausgewählten Commit und dem vorherigen kann eingesehen werden, indem die Datei ausgewählt wird.

Hinweis:

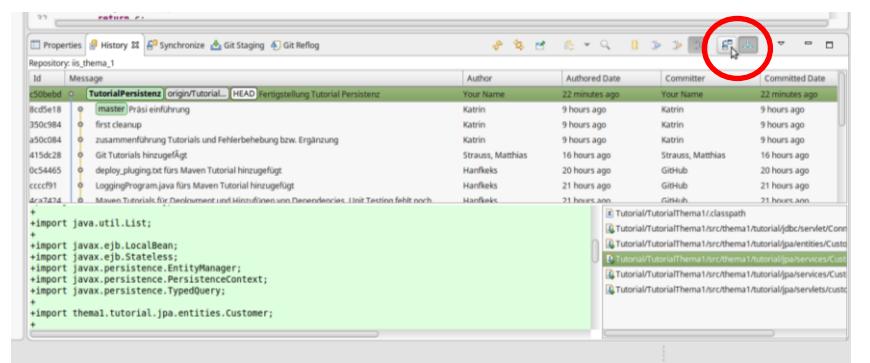
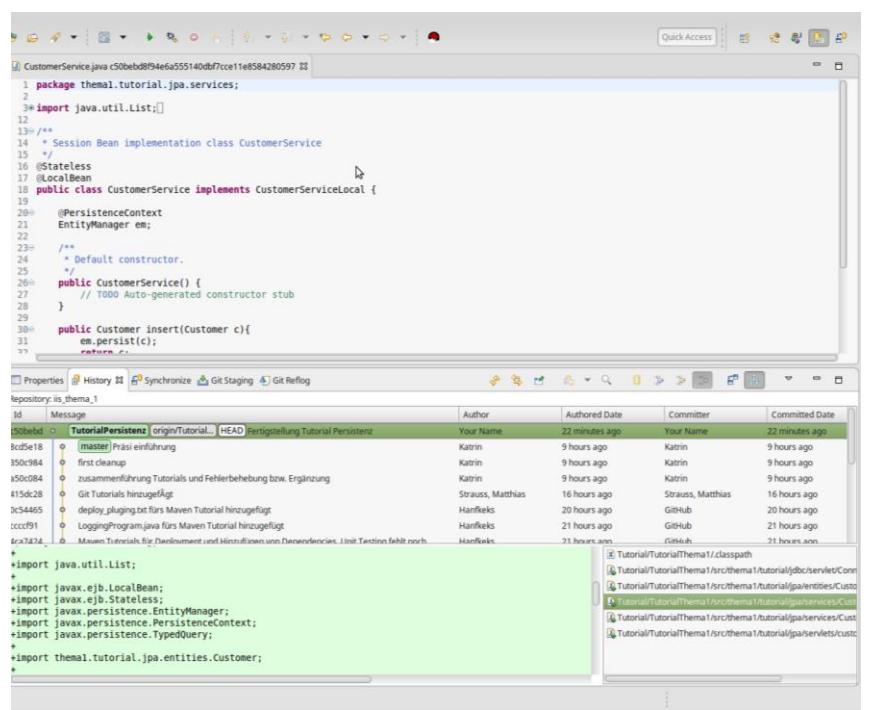
Auch die Git Kommandozeilen Befehle können einen Diff anzeigen. Der Befehl dafür lautet:

```
git diff
<commit>..<commit>
<filename>
```

4. Die vollständige Version der Datei kann per Doppelklick eingesehen werden.

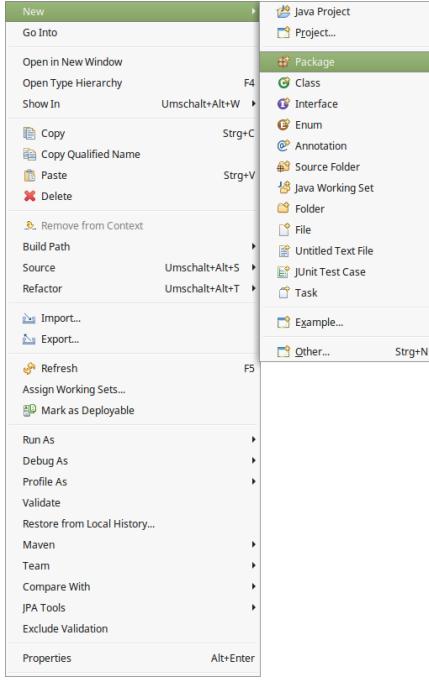
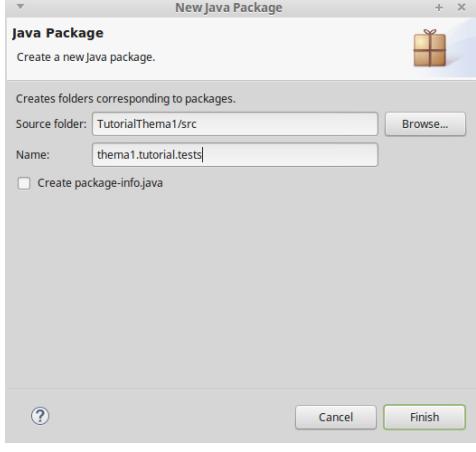


5. Um stattdessen eine andere Diff Ansicht zu bekommen kann der „Compare Mode“ aktiviert werden

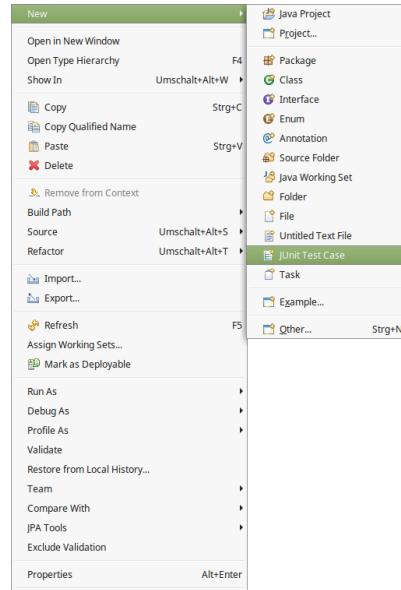


TEIL 16: UNIT TESTS

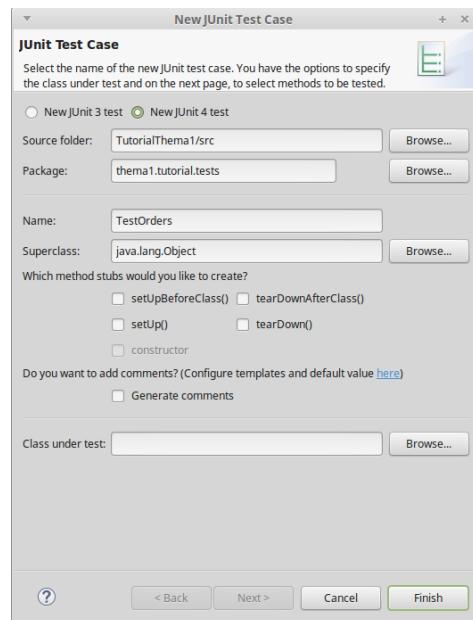
Im folgenden Abschnitt werden Unit Tests zum JPA Projekt hinzugefügt

Explanation	Screenshot
<ol style="list-style-type: none">1. Rechtsklick auf den src Ordner im Package Explorer → New → Package	
<ol style="list-style-type: none">2. Im Dialog den Namen auf thema1.tutorial.tests setzen und mit Finish bestätigen	

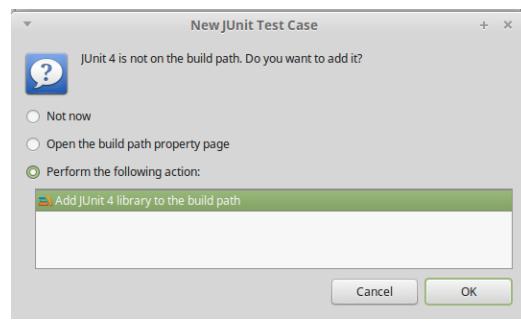
3. Rechtsklick auf das neue Package:
 → New
 → JUnit Test Case



4. Im Dialog den Namen auf TestOrders setzen und mit Finish bestätigen



5. Wenn ein Dialog auftaucht, die JUnit 4 lib hinzufügen und mit OK bestätigen



6. Code mit dem Inhalt der Datei TestOrders.java ersetzen

```

1 package thema1.tutorial.tests;
2
3 import static org.junit.Assert.*;
4
5 public class TestOrders {
6
7     @Test
8     public void testEquals() {
9         Orders o1 = new Orders();
10        Orders o2 = new Orders();
11        o1.setId(1);
12        o2.setId(1);
13        assertEquals(o1,o2);
14    }
15
16    @Test
17    public void testHash() {
18        Orders o1 = new Orders();
19        Orders o2 = new Orders();
20        o1.setId(1);
21        o2.setId(1);
22        assertEquals(o1.hashCode(),o2.hashCode());
23    }
24
25 }
26
27
28
29
30
31

```

7. Wie in Schritt 3-5 die JUnit Test Klasse TestCustomer hinzufügen



8. Die Inhalte der Datei TestCustomer.java in die entsprechende Datei in Eclipse einfügen

Im Code läuft kein kompilizierter Integrationstest sondern lediglich ein Test der Hash, ToString und Equals Methoden ab. Für einen Integrationstest muss eine EntityManagerFactory erstellt werden.

Dazu mal hier rein schaun:
<https://memorynotfound.com/unit-test-jpa-junit-in-memory-h2-database/>
 (muss allerdings ordentlich modifiziert werden)

```

public class TestCustomer {

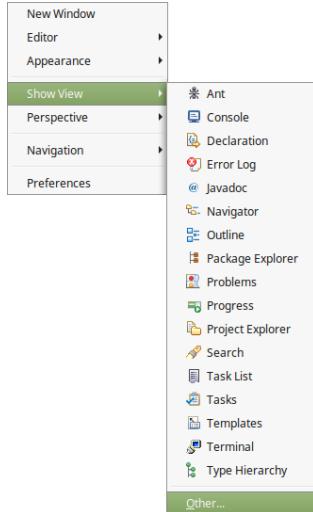
    @Test
    public void testEquals() {
        Customer c1 = new Customer();
        Customer c2 = new Customer();
        c1.setId(1);
        c2.setId(1);
        assertEquals(c1,c2);
    }

    @Test
    public void testHash() {
        Customer c1 = new Customer();
        Customer c2 = new Customer();
        c1.setId(1);
        c2.setId(1);
        assertEquals(c1.hashCode(),c2.hashCode());
    }

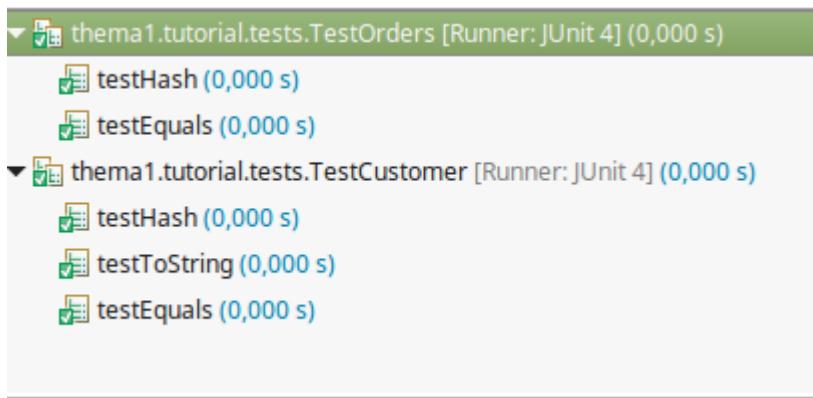
    @Test
    public void testToString() {
        Customer c1 = new Customer("first","last","email@address");
        Customer c2 = new Customer("first","last","email@address");
        c1.setId(1);
        c2.setId(1);
        assertEquals(c1.toString(),c2.toString());
    }
}

```

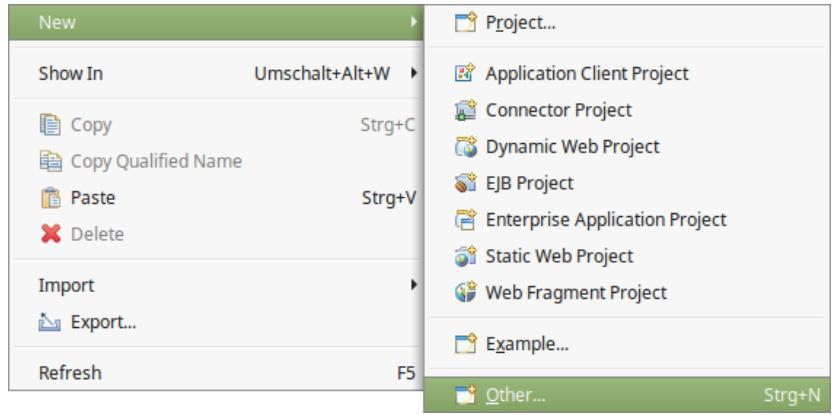
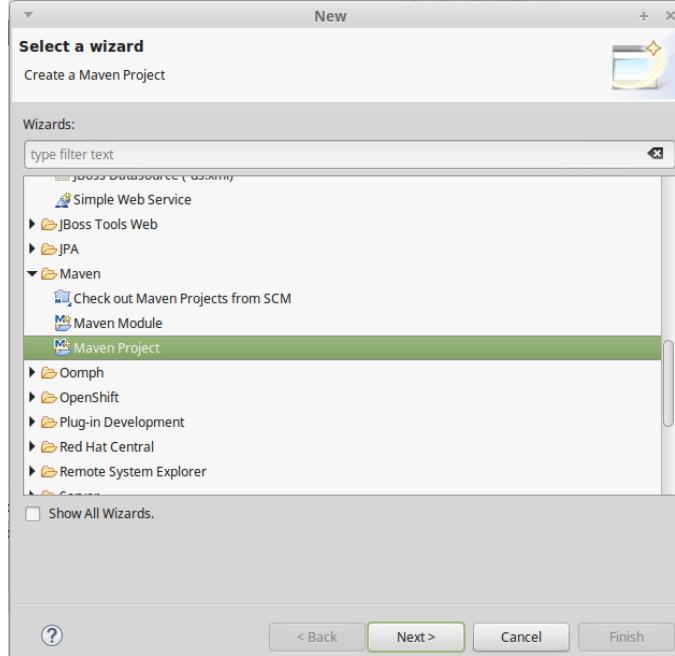
9. Über Window → Show View
→JUnit (evtl über Other...)
die JUnit Ansicht öffnen



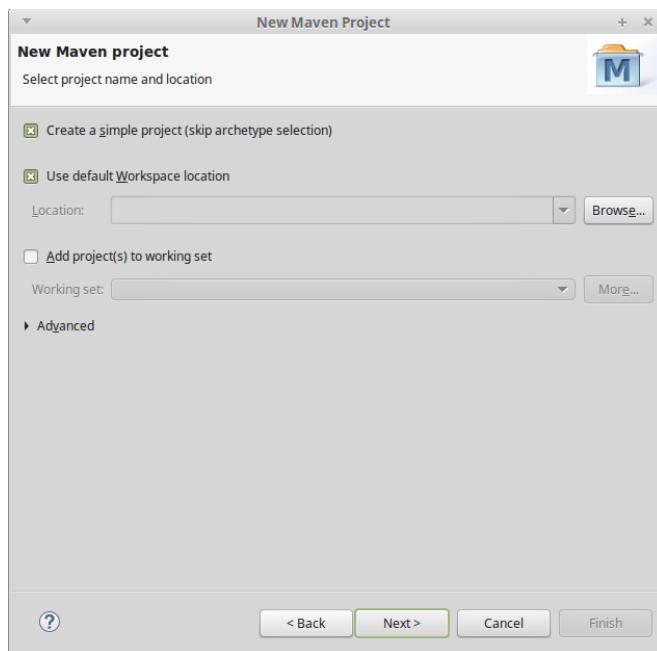
10. Rechtsklick auf
thema1.tutorial.tests Package
und Run As → JUnit Test
ausführen. Das Ergebnis sollte
wie folgt aussehen



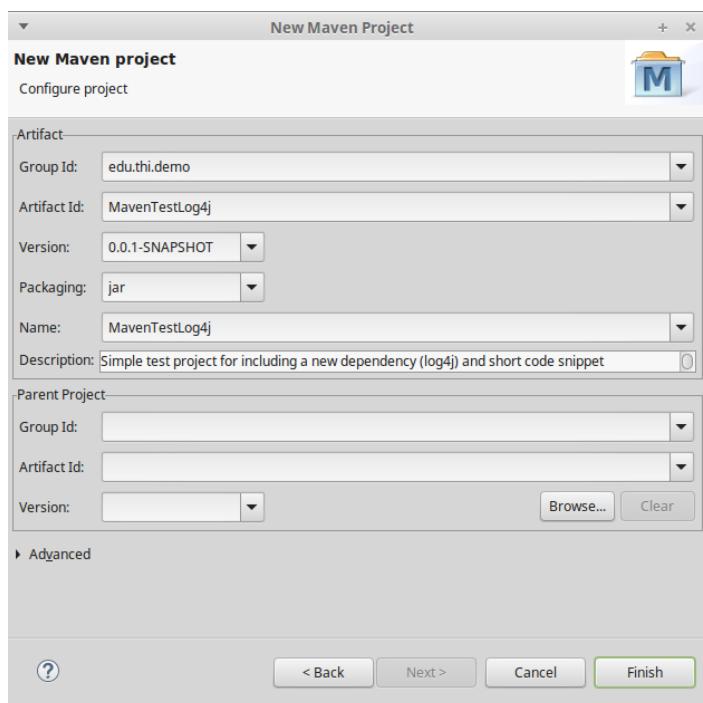
TEIL 17: PROJEKT MIT EINER NEUEN DEPENDENCY (HIER LOG4J)

Explanation	Screenshot
2. Rechtsklick in den Project Explorer → New → Other...	 A screenshot of the Eclipse 'New' context menu. The 'Other...' option is highlighted with a green box. Other options include 'Project...', 'Application Client Project', 'Connector Project', 'Dynamic Web Project', 'EJB Project', 'Enterprise Application Project', 'Static Web Project', 'Web Fragment Project', 'Example...', and 'Other...'. Shortcuts like 'Umschalt+Alt+W' for 'Show In' and 'Strg+C' for 'Copy' are visible.
3. Auswahl im Wizzard: Maven Project → Next	 A screenshot of the 'Select a wizard' dialog in Eclipse. The 'Maven Project' option is selected and highlighted with a green box. Other options include 'Simple Web Service', 'JBoss Tools Web', 'JPA', 'Check out Maven Projects from SCM', 'Maven Module', 'Oomph', 'OpenShift', 'Plug-in Development', 'Red Hat Central', and 'Remote System Explorer'. A 'Next >' button is visible at the bottom.

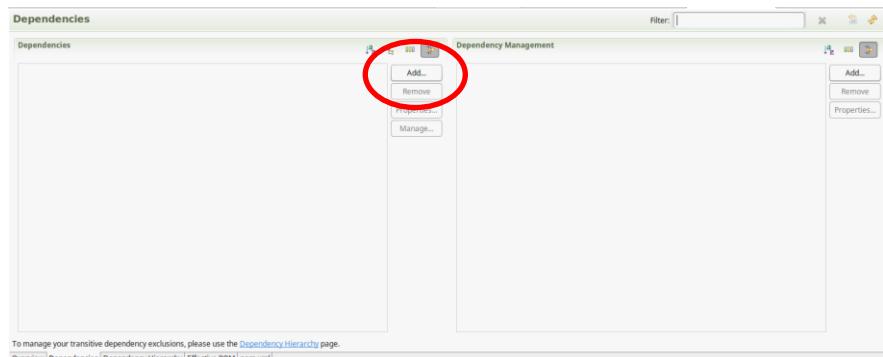
4. Im nächsten Schritt „Create a simple project (skip archetype selection)“ anwählen
 → Next



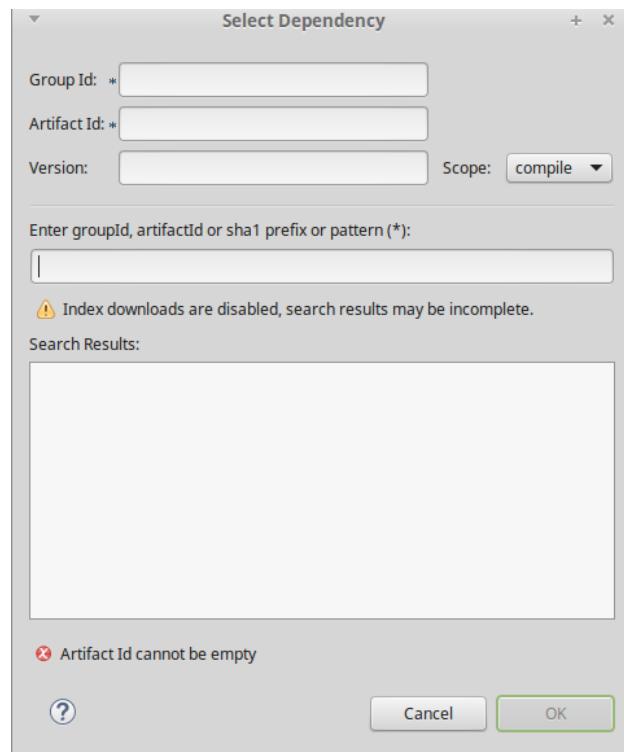
5. Group ID setzen:
 "edu.thi.demo"
 Artifact ID und Namen auf den
 selben Wert setzen
 Parent Project Sektion leer
 lassen.
 Optional eine Beschreibung
 Einfügen.
 → Finish



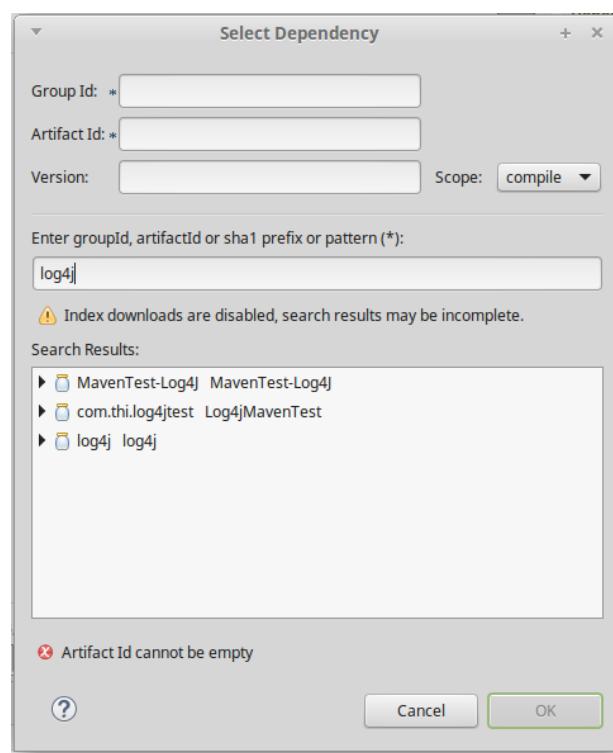
6. Doppelklick auf pom.xml
 Bei den Tabs im unteren
 Bereich des Fensters auf
 Dependencies wechseln
Anmerkung: Dieser Abschnitt
 tendiert dazu visuelle glitches
 zu verursachen.
 Wenn nötig kurz zurück auf
 Overview wechseln, dann
 zurück auf Dependencies



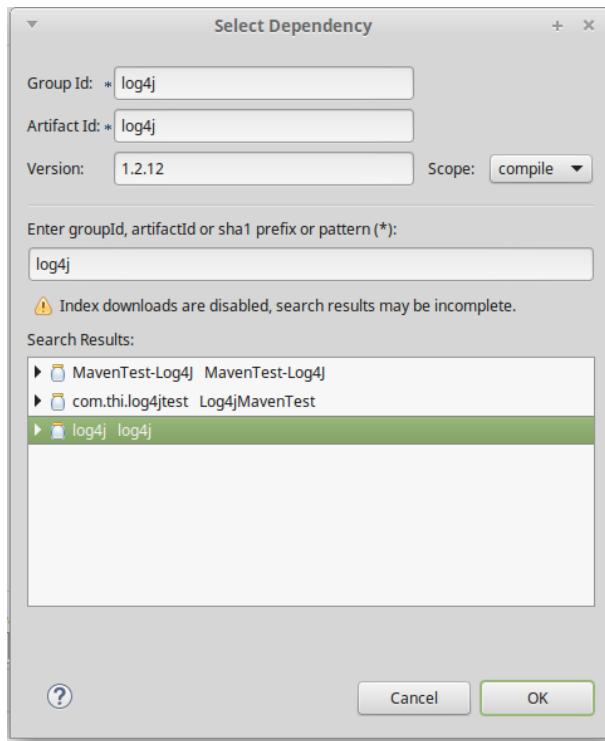
7. Im linken Dependencies Abschnitt auf „Add“ klicken, es öffnet sich folgender Dialog



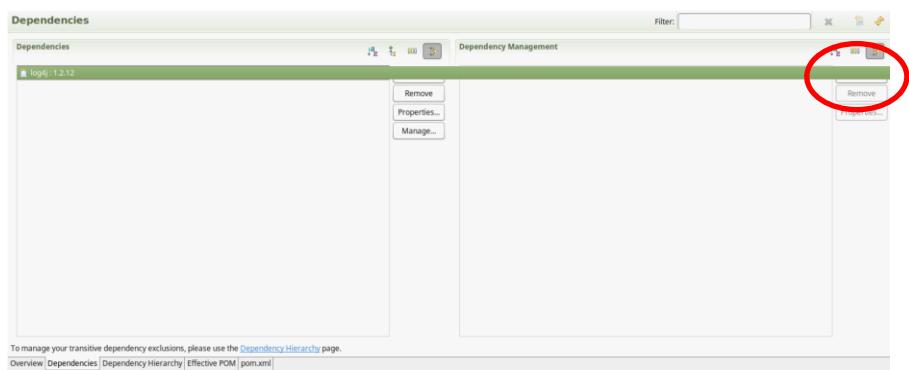
8. In dem Suchfenster „log4j“ eingeben, nicht mit Enter bestätigen sondern einfach kurz warten



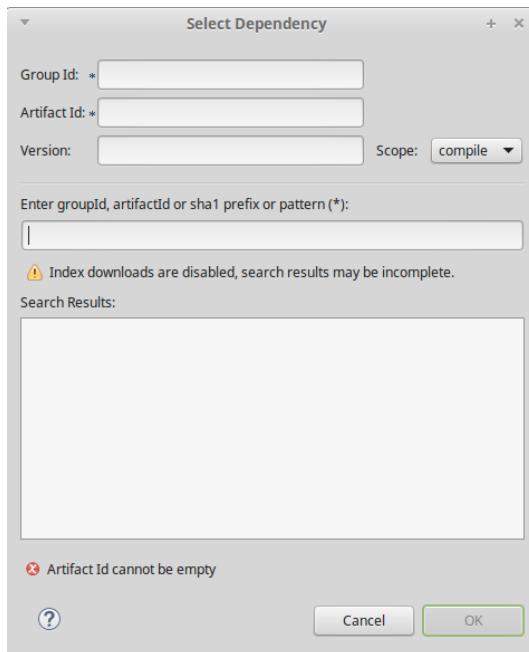
9. log4j mit einfachem klick auswählen und nachdem die Text Felder automatisch befüllt wurden auf ok drücken



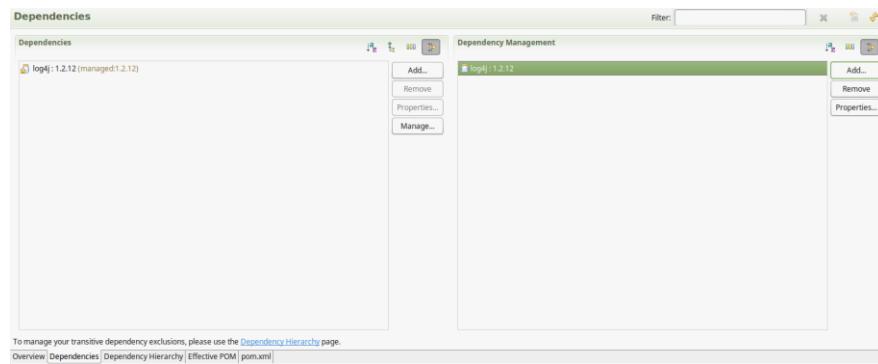
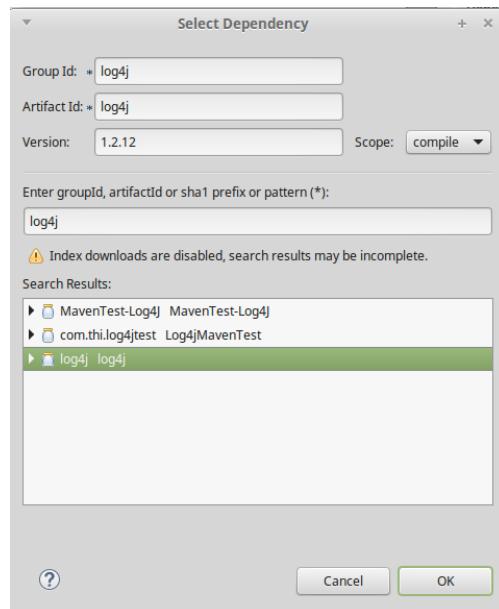
10. Möglicherweise tritt jetzt wieder ein visueller Glitch auf, welcher den im nächsten Schritt benötigten „Add“ Button verdeckt: Kurz auf Overview, dann zurück auf Dependencies wechseln



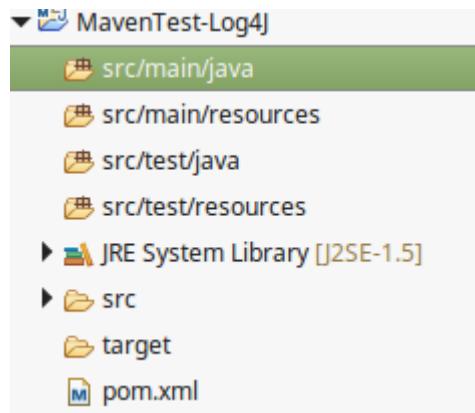
11. Nun rechts unter Dependency Management auf Add drücken, es öffnet sich wieder ein ähnlicher Dialog



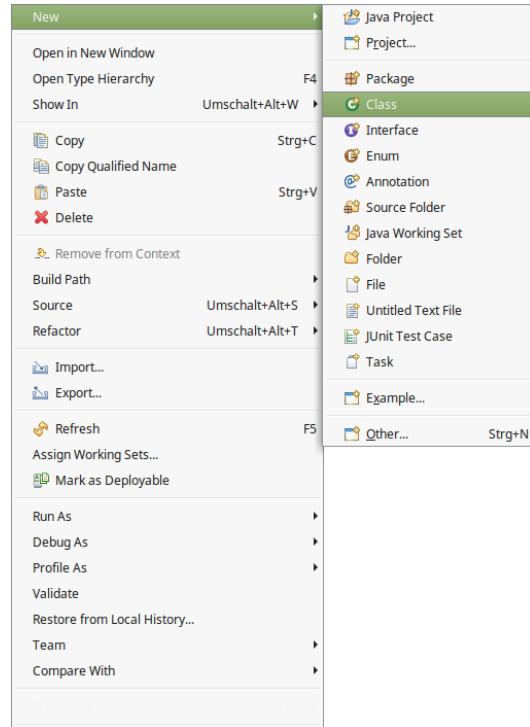
12. Wie oben log4j über die Suche auswählen
→ ok



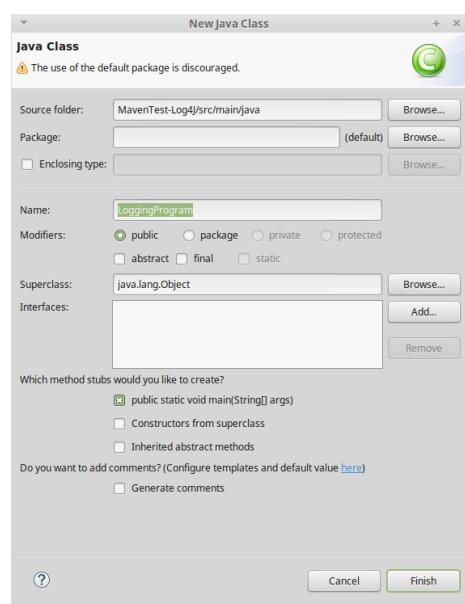
13. In den Package Explorer wechseln und Rechtsklick auf src/main/java



14. → New
→ Class



15. Als Namen LoggingProgram
eingeben
→ Finish



16. Code mit dem Inhalt von LoggingProgram.java aus Moodle ersetzen

Anmerkung: Nicht in Panik geraten wenn IntelliJ die Imports noch nicht kennt.

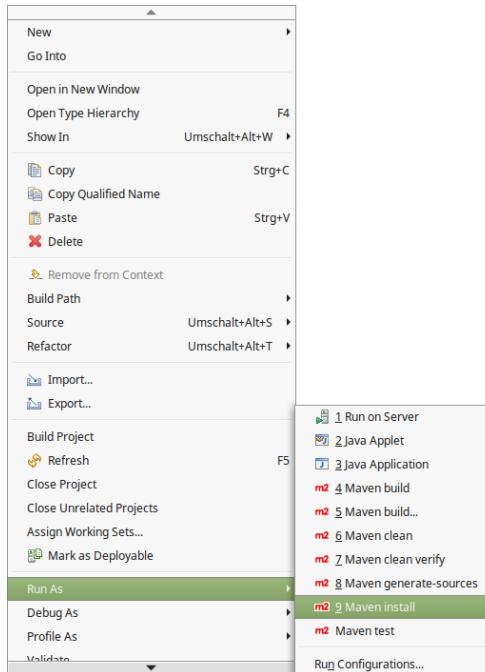
```

2 import org.apache.log4j.FileAppender;
3 import org.apache.log4j.Level;
4 import org.apache.log4j.Logger;
5 import org.apache.log4j.SimpleLayout;
6
7 public class LoggingProgram {
8
9     private static Logger logger = Logger.getRootLogger();
10
11    public static void main(String[] args)
12    {
13
14        try {
15            SimpleLayout layout = new SimpleLayout();
16            ConsoleAppender consoleAppender = new ConsoleAppender( layout );
17            logger.addAppender( consoleAppender );
18            FileAppender fileAppender = new FileAppender( layout, "logs/MeineLogDatei.log", false );
19            logger.addAppender( fileAppender );
20            // ALL | DEBUG | INFO | WARN | ERROR | FATAL | OFF:
21            logger.setLevel( Level.WARN );
22        } catch( Exception ex ) {
23            System.out.println( ex );
24        }
25        logger.debug( "Meine Debug-Meldung" );
26        logger.info( "Meine Info-Meldung" );
27        logger.warn( "Meine Warn-Meldung" );
28        logger.error( "Meine Error-Meldung" );
29        logger.fatal( "Meine Fatal-Meldung" );
30
31    }
32
33 }

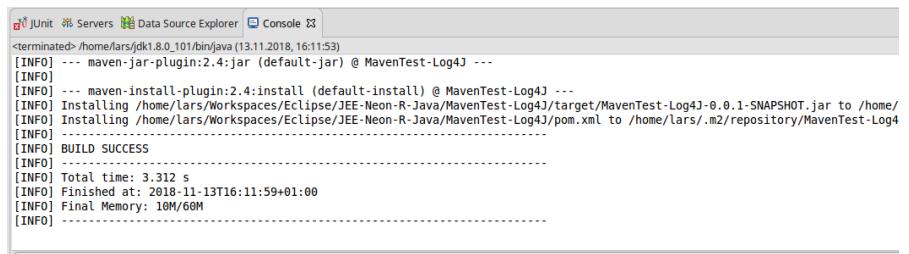
```

17. Strg + Shift + S drücken um alle Änderungen zu speichern!

Rechtsklick auf das Projekt → Run As → Maven install

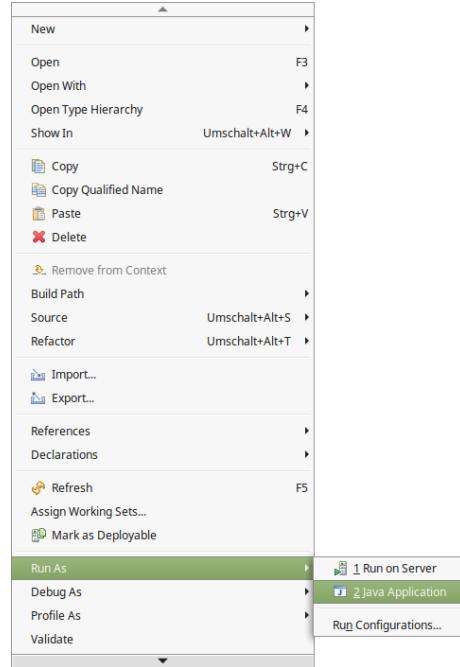


18. Im Konsolenfenster sollte nun BUILD SUCCESS erscheinen

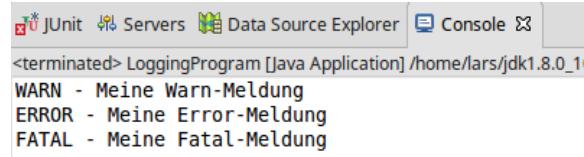


19. Rechtsklick auf LoggingProgram.java
→ Run As
→ Java Application
Anmerkung: Falls ein Dialog erscheint ob ungespeicherte Änderungen gespeichert werden sollen, auf Ok drücken und dann nochmal Maven install ausführen, da die ungespeicherten Änderungen nicht mitkompiliert wurden

Anmerkung 2: Falls Auto-Build deaktiviert ist, wird das Eclipse Projekt noch nicht auf dem neusten Stand sein und immernoch angezeigt, dass die Dependencies fehlen. Dann einfach nochmal Build Project ausführen



20. Ausgabe sollte folgendermaßen aussehen:



```
<terminated> LoggingProgram [Java Application] /home/lars/jdk1.8.0_11
WARN - Meine Warn-Meldung
ERROR - Meine Error-Meldung
FATAL - Meine Fatal-Meldung
```

TEIL 18: MAVEN AUTOMATISIERTES DEPLOYEN

Explanation	Screenshot
<p>7. Im Projekt orderhandling-wildfly-process die pom.xml öffnen und in die XML Ansicht wechseln</p>	<p>1 <?xml version="1.0" encoding="UTF-8"?></p> <p>2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" modelVersion="4.0.0"></p> <p>3 <groupId>edu.thi.demo</groupId></p> <p>4 <artifactId>orderhandling-wildfly-process</artifactId></p> <p>5 <version>0.0.1-SNAPSHOT</version></p> <p>6 <packaging>war</packaging></p> <p>7 <name>Camunda BPM Process Application</name></p> <p>8 <description>A Process Application for [Camunda BPM] (http://docs.camunda.org/bpm-platform/7.7/process-applications/)</description></p> <p>9 <properties></p> <p>10 <camunda.version>7.7.0</camunda.version></p> <p>11 <!--</p> <p>12 Adjust if you want to use Camunda Enterprise Edition (EE):</p> <p>13 <camunda.version>7.9.0-ee</camunda.version></p> <p>14 Make sure you also switch to EE repository below</p> <p>15 --></p> <p>16 <maven.compiler.source>1.8</maven.compiler.source></p> <p>17 <maven.compiler.target>1.8</maven.compiler.target></p> <p>18 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding></p> <p>19 <failOnMissingWebXml>false</failOnMissingWebXml></p> <p>20 </properties></p> <p>21 <dependencyManagement></p> <p>22 <dependencies></p> <p>23 <dependency></p> <p>24 <groupId>org.camunda.bpm</groupId></p> <p>25 <artifactId>camunda-bom</artifactId></p> <p>26 <version>\${camunda.version}</version></p> <p>27 </dependency></p> <p>28 </dependencies></p> <p>29 </dependencyManagement></p> <p>30 <dependency></p> <p>31 <groupId>org.camunda.bpm</groupId></p> <p>32 <artifactId>camunda-bpm-process-app</artifactId></p> <p>33 <version>\${camunda.version}</version></p>
<p>8. Unter dem Tag <build> den <plugins> tag suchen und dort den Inhalt der Datei deploy_plugin.txt einfügen</p>	<pre> 180 <build> 181 <finalName>\${project.artifactId}</finalName> 182 <plugins> 183 184 <plugin> 185 <!-- Deploy to Tomcat using: mvn clean package antrun:run 186 Follow the instructions in build.properties.example to make it work!--> 187 <groupId>org.apache.maven.plugins</groupId> 188 <artifactId>maven-antrun-plugin</artifactId> 189 <configuration> 190 <tasks> 191 <ant antfile="\${basedir}/build.xml"> 192 <target name="copy.war.into.tomcat" /> 193 </ant> 194 </tasks> 195 </configuration> 196 </plugin> 197 <!-- Tomcat Maven Plugin 198 199 Deploy to Tomcat using: 200 mvn clean tomcat7:deploy 201 202 Redeploy: 203 mvn clean tomcat7:redeploy 204 205 Undeploy: 206 mvn tomcat7:undeploy </pre>
<p>9. Überprüfen ob die Syntax korrekt ist und der Inhalt der Datei (beginnend und endend mit dem <plugin> tag) richtig im eingefügt wurde.</p> <p>Anmerkung: Da in unserem Projekt alles auf einer Maschine läuft sind keine Uploads nötig. Für das simple Kopieren gibt es mehrere Möglichkeiten und verschiedene Plugins. In diesem StackOverflow Post sind einige davon aufgelistet:</p>	<pre> <finalName>\${project.artifactId}</finalName> <plugins> <plugin> <artifactId>maven-resources-plugin</artifactId> <version>2.7</version> <executions> <execution> <id>copy-file</id> <phase>install</phase> <goals> <goal>copy-resources</goal> </goals> <configuration> <resources> <resource> <directory>/home/lars/Workspaces/Eclipse/JEE-Neon-R-Java/orderhandling-wildfly-process/target</directory> <includes> <include>orderhandling-wildfly-process.war</include> </includes> </resource> </resources> <outputDirectory>/home/lars/camunda/server/wildfly-10.1.0.Final/standalone/deployments</outputDirectory> </configuration> </execution> </executions> </plugin> </pre>

<https://stackoverflow.com/questions/586202/best-practices-for-copying-files-with-maven>

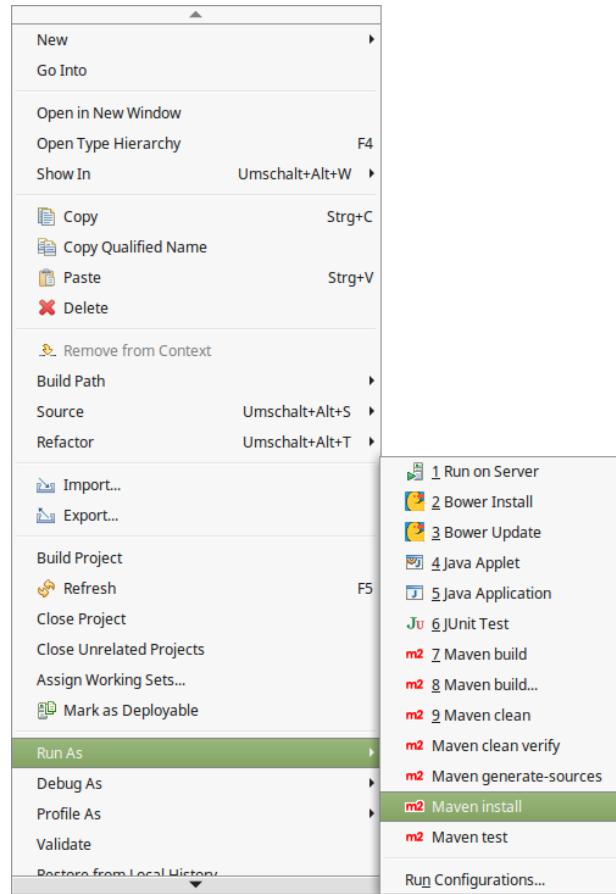
Wer also stattdessen AntRun nutzen möchte, kann den Schritten dort folgen.
Eine AntRun dependency sollte bereits in der pom vorhanden sein

10. Alles speichern mit Strg + Shift + S

Maven install durch Rechtsklick auf das Projekt → Run As → Maven install ausführen

Erklärung:

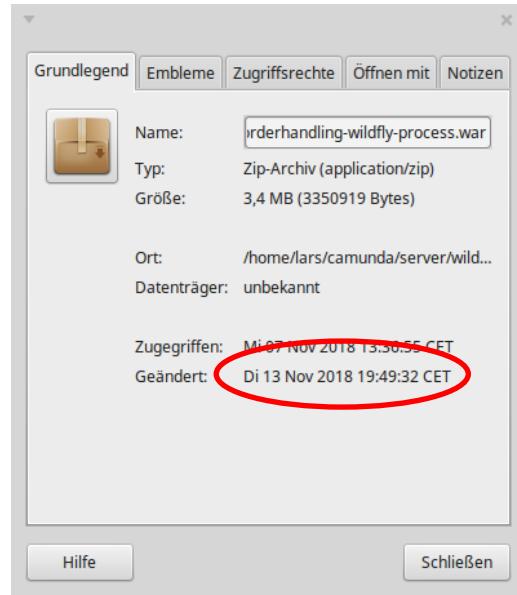
Obwohl es um Deployment geht, nutzen wir nicht die Phase mvn deploy. Der Grund dafür ist, dass es sich bei deploy um das Hochladen von Dateien auf Remotes dreht.
Unser Deployment ist lediglich das Kopieren der fertigen Datei in einen anderen Ordner



11. Im Ordner

/home/lars/camunda/server/wildfly-10.1.0.Final/standalone/deployments

auf die Datei orderhandling-wildfly-process.war rechtsklicken > Eigenschaften und das Änderungsdatum beachten. Wenn das Änderungsdatum identisch mit der jetzigen Zeit ist, wurde die Datei kopiert.



ANHANG: JDBC ZUSATZSERVLETS

In diesem Abschnitt wird erklärt, wie die Servlets CreateAdresse, SelectAdresse, UpdateAdresse und DeleteAdresse ausgeführt werden können, was dabei berücksichtigt werden muss, wie die Ergebnisse in der Datenbank überprüft werden können und welche Ausgabe erwartet wird.

Innerhalb der Servlets gibt es die identische Funktion zweimal: Dabei wird sie einmal mit einem nativen SQL-String implementiert und einmal mit einem PreparedStatement, wobei hier Parameter innerhalb des SQL-Queries durch Werte ersetzt werden können und so eine Wiederverwendung des Queries möglich ist.

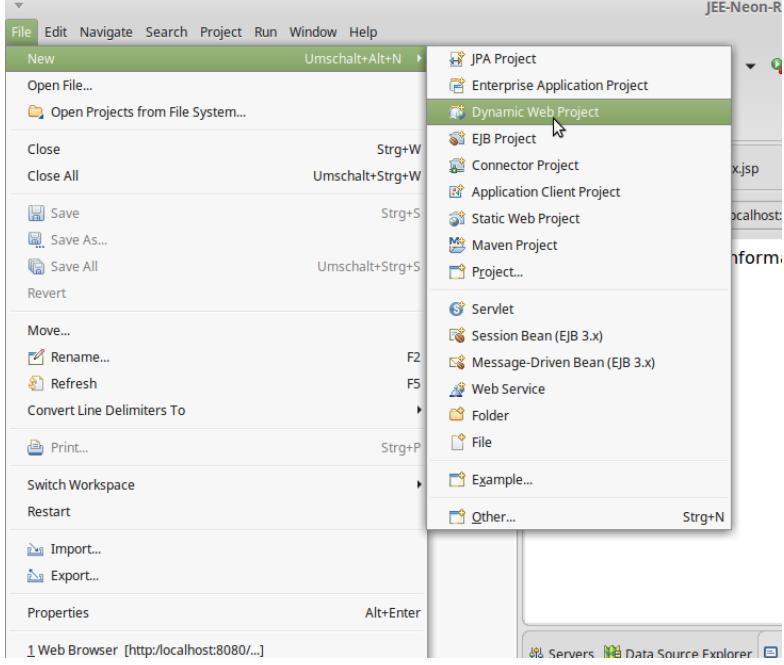
Voraussetzung für die Ausführung dieses Teils ist es, dass die MySQL-Datenbank gestartet ist und der WildFly-Server fehlerfrei läuft.

Explanation	Screenshot
6. Um diese Servlets auszuführen, muss einmalig ein Build und ein Publish erfolgen (Schritt 6 und 7 in Teil X), bevor die Servlets ausgeführt werden können (analog zu den Schritten 8 und 9 in Teil X).	
7. CreateAdresse: Es werden zwei Customer in der Datenbank angelegt. Das Servlet kann unabhängig von anderen Servlets gestartet werden. Die erwartete Ausgabe zeigt das Bild rechts.	<pre>Insert plain SQL Try to insert: INSERT INTO Adresse VALUES(0, 'Ingolstadt', '5', '85055', 'Esplanade') inserted successfully Insert Prepared Statement: INSERT INTO Adresse VALUES (?, ?, ?, ?, ?) inserted successfully</pre>
8. Das Ergebnis kann auch in der Datenbank überprüft werden. Im Terminal: mysql -u root -p Passwort = master42	<pre>lars@ikm00-ws16:~\$ mysql -u root -p Enter password: Welcome to the MariaDB monitor. Commands end with ; or \g. Your MariaDB connection id is 32 Server version: 5.5.49-MariaDB-1ubuntu0.14.04.1 (Ubuntu) Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others. Type 'help;' or '\h' for help. Type '\c' to clear the current input statement. MariaDB [(none)]></pre>
9. Anschließend die Datenbank shop auswählen: use shop;	<pre>MariaDB [(none)]> use shop; Reading table information for completion of table and column names You can turn off this feature to get a quicker startup with -A Database changed MariaDB [shop]></pre>
10. Beim selektieren aller Zeilen in der Tabelle Adresse sollten zwei Einträge angezeigt werden: select * from Adresse;	<pre>MariaDB [shop]> select * from Adresse; +-----+-----+-----+-----+ id city houseNumber postalCode street +-----+-----+-----+-----+ 0 Ingolstadt 5 85055 Esplanade 1 München 116 809 Frankfurter Ring +-----+-----+-----+-----+ 2 rows in set (0.00 sec) MariaDB [shop]></pre>

<p>11. UpdateAdresse: Voraussetzung ist, dass das Servlet CreateAdresse zuvor ausgeführt wurde, damit die Datenbank entsprechende Einträge enthält. Dieses Servlet ändert beide Einträge in der Datenbank. Die erwartet Ausgabe zeigt das Bild rechts.</p>	<pre>Insert plain SQL Try to update: UPDATE Adresse SET houseNumber = 10 WHERE id = 0 Result: 0 Ingolstadt 10 85055 updated successfully Update with Prepared Statement: UPDATE Adresse SET houseNumber = ? WHERE id = ? Result: 1 München 100 809</pre>
<p>12. Auch dieses Ergebnis kann in der Datenbank kontrolliert werden, indem die Schritte 3 und 4 ausgeführt wurden und anschließend Schritt 5 erneut ausgeführt wird.</p>	<pre>MariaDB [shop]> select * from Adresse; +----+-----+-----+-----+ id city houseNumber postalCode street +----+-----+-----+-----+ 0 Ingolstadt 10 85055 Esplanade 1 München 100 809 Frankfurter Ring +----+-----+-----+-----+ 2 rows in set (0.00 sec) MariaDB [shop]></pre>
<p>13. DeleteAdresse: Voraussetzung ist, dass das Servlet CreateAdresse zuvor ausgeführt wurde, damit die Datenbank entsprechende Einträge enthält. Dieses Servlet löscht beide Datensätze. Die erwartet Ausgabe zeigt das Bild rechts.</p>	<pre>Delete with plain SQL Try to delete: DELETE FROM Adresse WHERE id = 0 Result: 1 München 116 809 deleted successfully Delete with Prepared Statement: DELETE FROM Adresse WHERE id = ? Result:</pre>
<p>14. Auch dieses Ergebnis kann in der Datenbank kontrolliert werden, indem die Schritte 3 und 4 ausgeführt wurden und anschließend Schritt 5 erneut ausgeführt wird.</p>	<pre>MariaDB [shop]> select * from Adresse; Empty set (0.00 sec) MariaDB [shop]></pre>
<p>15. SelectAdresse: Voraussetzung ist, dass das Servlet CreateAdresse zuvor ausgeführt wurde, damit die Datenbank entsprechende Einträge enthält. Dieses Servlet selektiert den Datensatz, bei dem die Id übereinstimmt. Die erwartet Ausgabe zeigt das Bild rechts. Beide Einträge wurden gelöscht.</p>	<pre>Connection erfolgreich: true Select with plain SQL Try to select: SELECT * FROM Adresse WHERE id = 0 Result: 0 Ingolstadt 5 85055 Select with Prepared Statement: SELECT * FROM Adresse WHERE id=? Result: 1 München 116 809</pre>

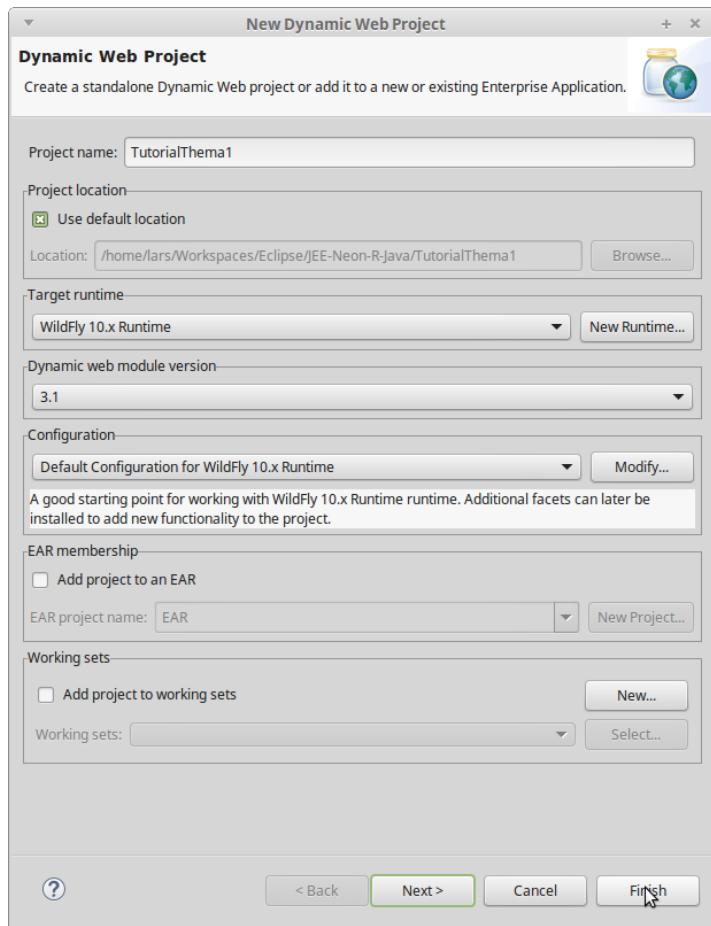
ANHANG: ECLIPSE PROJEKT ERSTELLEN

Im Folgenden wird erklärt, wie man ein neues Eclipse-Projekt erstellt, das als Webanwendung auf einem WildFly-Server läuft und JDBC, JPA und Maven nutzt.

Explanation	Screenshot
1. In Eclipse-Menü <i>File</i> > <i>New</i> > <i>Dynamic Web Project</i> wählen.	 A screenshot of the Eclipse IDE interface. The window title is 'JEE-Neon-R'. The 'File' menu is open, showing various options like 'New', 'Open File...', 'Save', and 'Properties'. A sub-menu is displayed under 'New', listing project types: 'JPA Project', 'Enterprise Application Project', 'Dynamic Web Project', 'EJB Project', 'Connector Project', 'Application Client Project', 'Static Web Project', 'Maven Project', 'Project...', 'Servlet', 'Session Bean (EJB 3.x)', 'Message-Driven Bean (EJB 3.x)', 'Web Service', 'Folder', 'File', 'Example...', and 'Other...'. The 'Dynamic Web Project' option is highlighted with a green background and a white border, indicating it is selected.

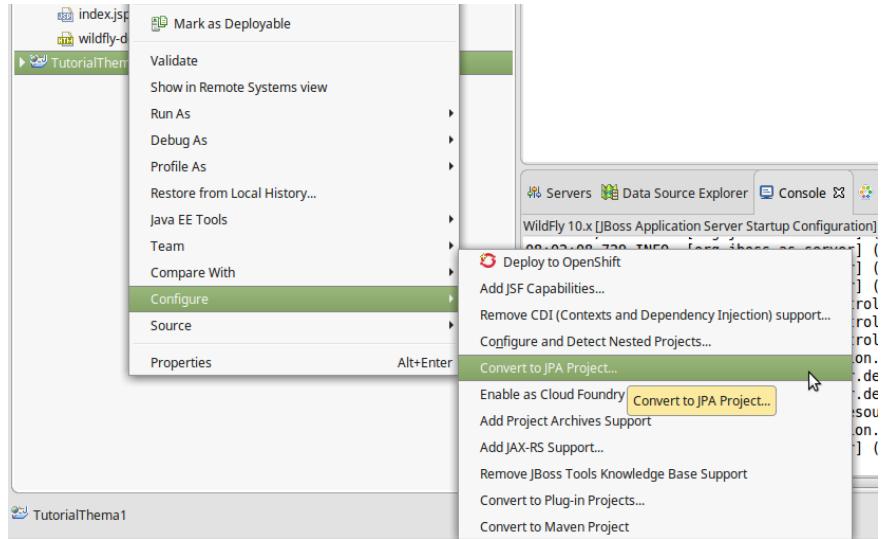
2. Anschließend den Projektnamen eingeben und prüfen, dass alle Einstellungen den nebenstehenden entsprechen.

Mit *Finish* beenden.



3. Da wir in unserem Web-Projekt JPA nutzen wollen, können wir das Projekt direkt in ein JPA-Projekt konvertieren.

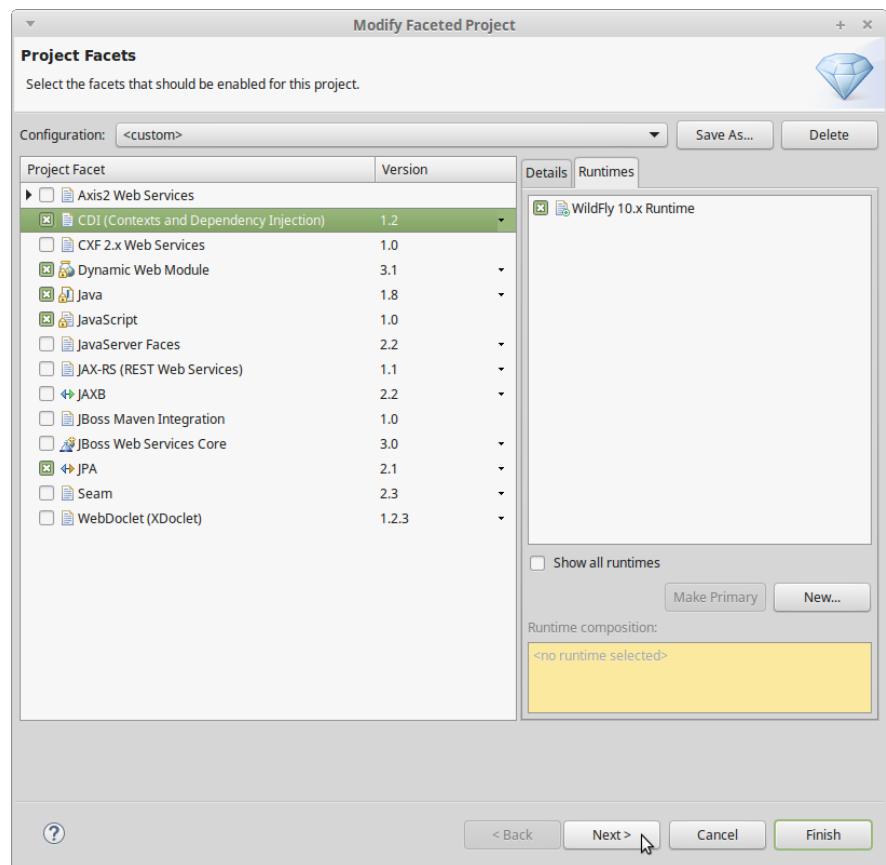
Rechtsklick auf den Projektordner > Configure > Convert to JPA Project...



4. Bei den Einstellungen alle Kreuzchen so setzen wie im nebenstehenden Bild.

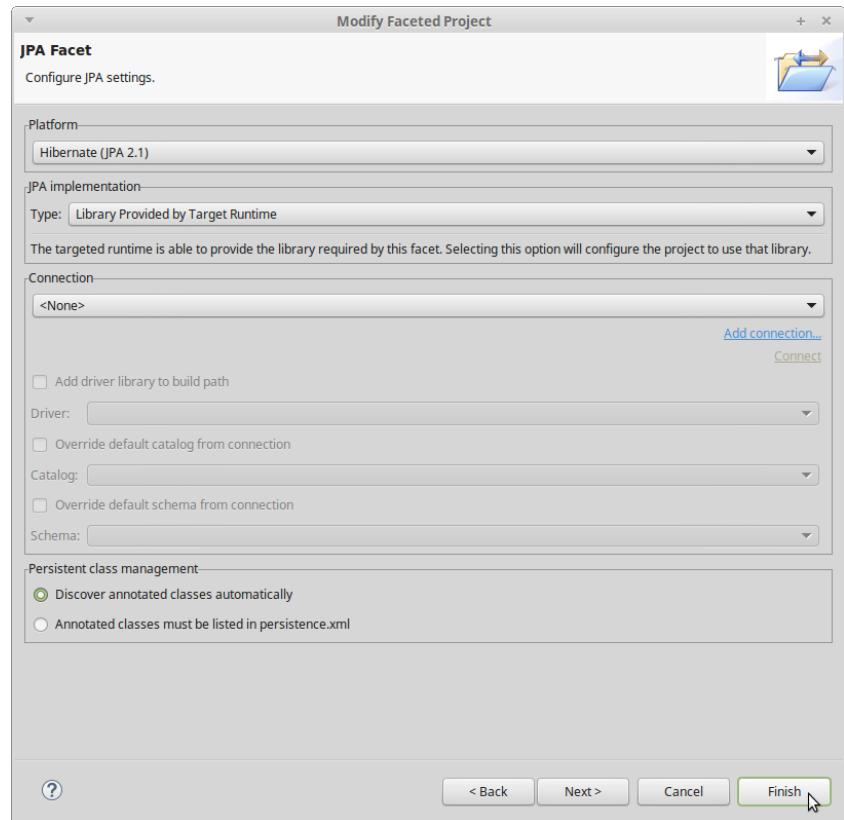
Außerdem darauf achten, dass rechts im Reiter **Runtimes** WildFly ausgewählt ist.

Mit *Next* zum nächsten Schritt navigieren.



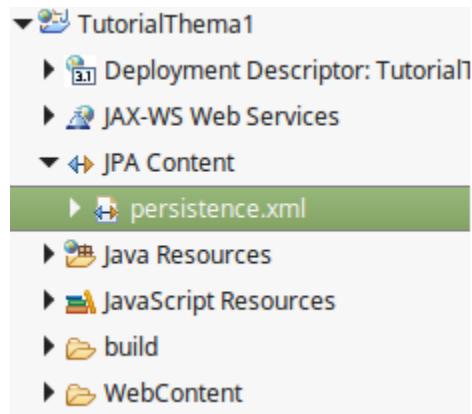
5. Hier den Type der JPA Implementation auf *Library Provided by Target Runtime* setzen.

Mit *Finish* beenden.



6. Nun müssen in der Datei persistence.xml einige Änderungen vorgenommen werden. Dazu die Datei öffnen.

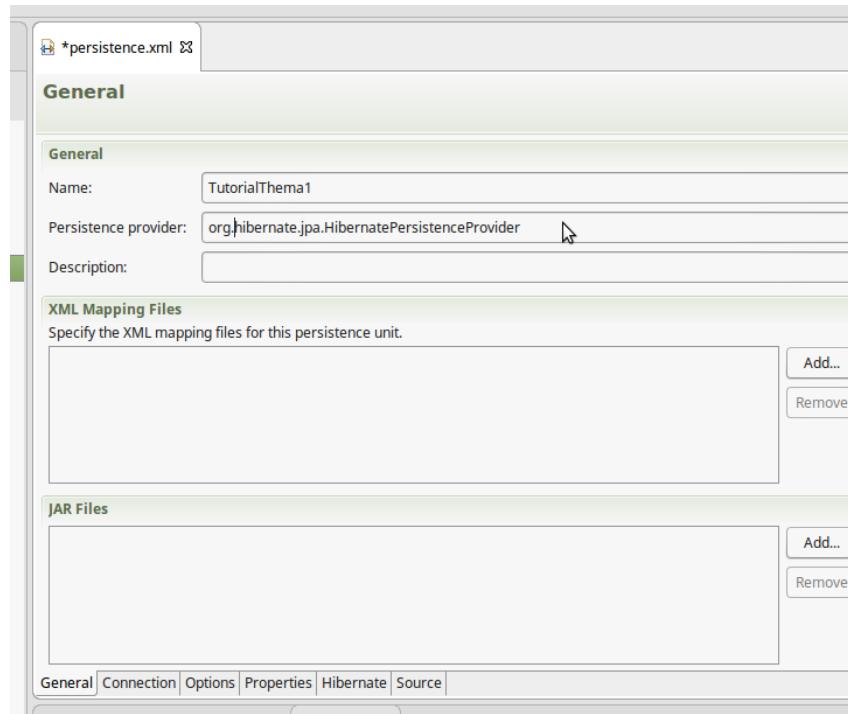
Hintergrund:
In dieser Datei werden alle Einstellungen für eine Persistenzeinheit gespeichert. Dazu gehört unter Anderem der Provider, sowie die Datenbank.



7. Im Reiter General wird der Provider festgelegt.

In das Feld **Persistence provider** folgendes eintragen: `org.hibernate.jpa.HibernatePersistenceProvider`

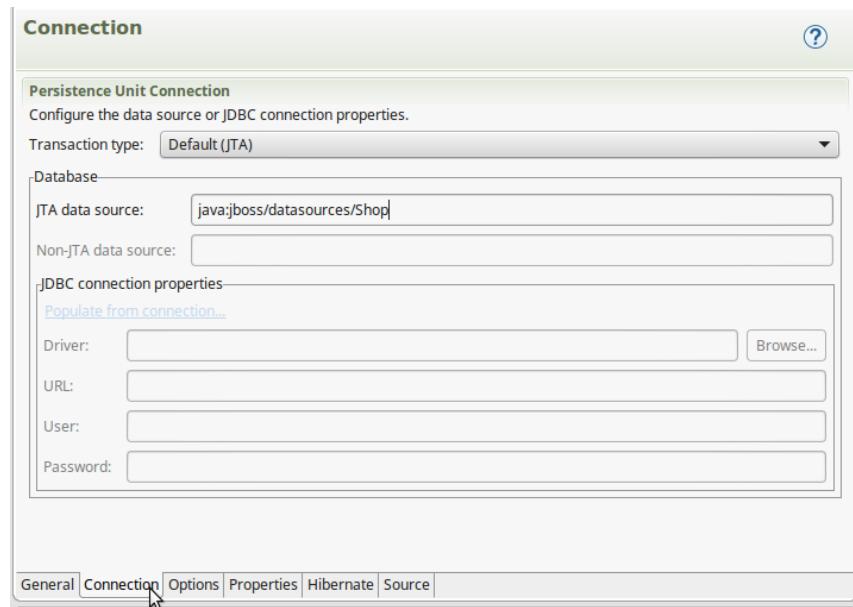
Hintergrund:
Wir nutzen hier Hibernate. Es gibt einige andere Möglichkeiten, wie beispielsweise EclipseLink.



8. Im Reiter Connection wird die JTA DataSource hinterlegt.

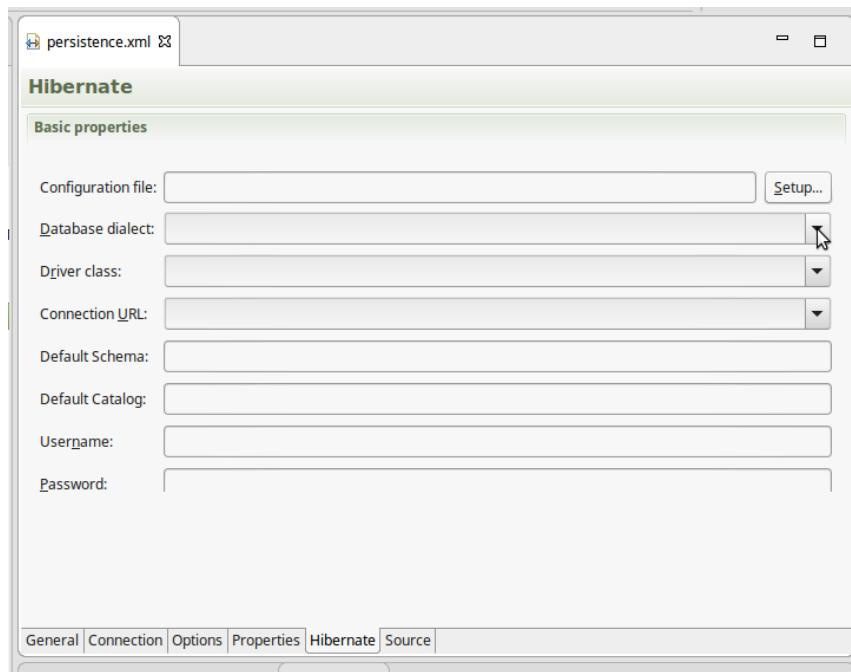
`java:jboss/datasource/Shop`

Der genaue Name lässt sich aus der für WildFly konfigurierten DataSource entnehmen. Dabei JNDI nutzen.



9. Nun zum Reiter **Hibernate** wechseln, um die Datenbank genauer zu spezifizieren.

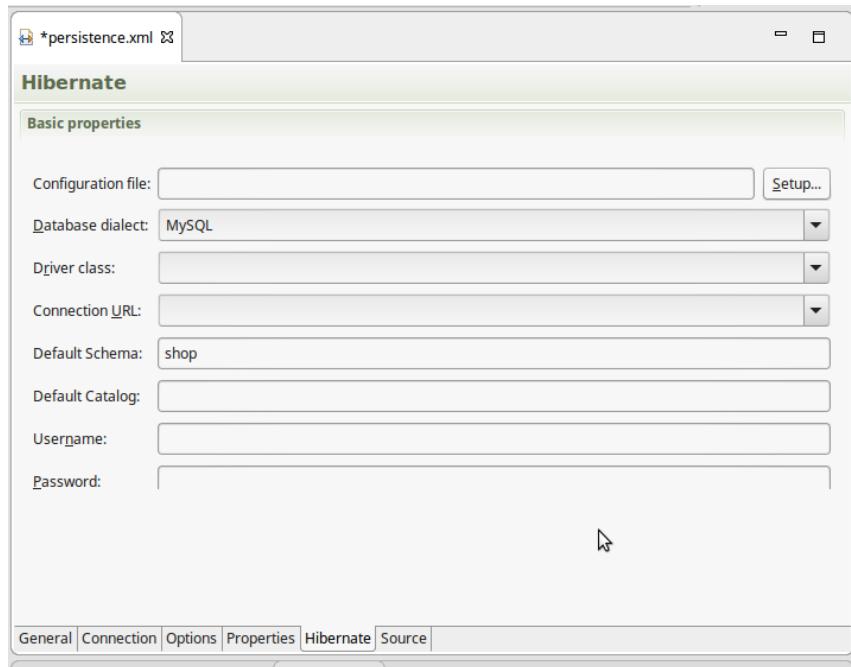
Im DropDown *Database dialect* wird MySQL ausgewählt.



10. Als Default Schema den Namen des zu nutzenden Schemas angeben.

Im Tutorial: shop

Änderungen speichern.



11. Im Reiter Properties sollten bereits zwei Eigenschaften aufgelistet werden.

Das dritte über den Add-Button hinzufügen.

Name:
hibernate.hbm2ddl.auto
Value: create-drop

Hintergrund:

Diese Eigenschaft haben wir im Tutorial genutzt, damit nicht extra alle Tabellen angelegt werden müssen. Denn ist das Attribut auf create-drop gesetzt, werden automatisch beim Start alle Tabellen zunächst gelöscht und dann neu erstellt. Dies ist jedoch nicht sinnvoll, wenn man seine Daten dauerhaft speichern möchte. In diesem Fall in der Dokumentation nachlesen, welche Alternativen es gibt!

Name	Value
hibernate.dialect	org.hibernate.dialect.MySQLDialect
hibernate.default_schema	shop
hibernate.hbm2ddl.auto	create-drop