Tutorial Git, Peristenz, Maven, Junit Tests

Teil 1: Einrichtung von git

|  |  |
| --- | --- |
| **Explanation** | **Screenshot** |
| 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:   *git config user.name <username>*  *bzw.*  *git config user.email <email>* |  |

Teil 2: Repository erstellen

|  |  |
| --- | --- |
| **Explanation** | **Screenshot** |
| 1. In das Verzeichnis navigieren, in dem das Repository angelegt werden soll. |  |
| 1. Git Repository initialisieren:   git init |  |
| 1. Das Repository ist einsatzbereit! |  |

Teil 3: Staging and Commiting

|  |  |
| --- | --- |
| **Explanation** | **Screenshot** |
| 1. Sobald Sie mit den Änderungen in Ihrem Arbeitsverzeichniss zufrieden sind können Sie eine Liste aller geänderten Dateien anzeigen lassen:   git status |  |
| 1. Nachdem eine neue Datei erstellt wurde ist sie zunächst „untracked“. Damit sie von Git verwaltet werden kann muss sie mit *git add <dateiname>* mit aufgenommen werden. („Staging“). 2. Nun zeigt *git status* einen neuen Zustand: „zum Commit vorgemerkte Änderungen“. Alle Änderungen der hier gelisteten Dateien werden beim nächsten Commit abgespeichert. |  |
| 1. Ein Commit wird mit dem Befehl *git commit -m <message>* durchgeführt. Dadurch werden die Änderungen der „staged“ Dateien abgespeichert. |  |
| 1. Falls alle Änderungen gestaged wurden, zeigt *git status* nach dem commit an, dass das repo clean ist und keine Änderungen vorliegen |  |
| 1. Dieses Vorgehen wird für jeden Commit wiederholt. |  |

Teil 4: Arbeiten mit branches

|  |  |
| --- | --- |
| **Explanation** | **Screenshot** |
| 1. Einen neuen Branch erstellen:   git branch <branchname> |  |
| 1. In den neuen Branch wechseln:   git checkout <branchname> |  |
| 1. Neue Änderungen machen und commiten |  |
| 1. In den master Branch zurückwechseln |  |
| 1. Den neuen Branch in den aktuellen mergen:   git merge <branchname> |  |
| 1. Alle Änderung, die im Branch javaHelloWorld gemacht wurden, sind jetzt auch in master. |  |

Teil 5: Github als Remote für ein neues projekt

|  |  |
| --- | --- |
| **Explanation** | **Screenshot** |
| 1. Anmeldung auf <https://github.com/login>   **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 |  |
| 1. Ein neues Repository anlegen |  |
| 1. Den Namen des Repositorys eintragen und „Create Repository“ klicken. |  |
| 1. Ein neues Projekt anlegen und ein git Repository mit git init initialisieren. In diesem Projekt die Readme anlegen:   echo "# RemoteTutorial" >> README.md |  |
| 1. Die Readme commiten |  |
| 1. Die URL des GitHub Repository kopieren |  |
| 1. Das GitHub Repository als Remote hinzufügen:     *git remote add origin <URL>* |  |
| 1. Den Status des Projektes in das GitHub Repository hochladen:   git push -u origin master  Wenn angefragt Benutzername und Passwort angeben. |  |
| 1. Auf GitHub ist nun der aktuelle Status zu sehen |  |

Teil 6: Ein bestehendes Repository clonen, Remote branches auschecken und merge-Konflikte lösen

|  |  |
| --- | --- |
| **Explanation** | **Screenshot** |
| 1. Klonen des Tutorial Repository:   git clone https://github.com/THI-Tutorial-User/RemoteTutorial.git |  |
| 1. In das Verzeichnis wechseln und ein erweitertes Log betrachten:   ﻿  git log --graph --all --decorate |  |
| 1. Erstellen eines lokalen Branches, der dem Remote Branch *origin/tutorialBranch* folgt und den neuen Branch auschecken:   git checkout -b tutorialBranch origin/tutorialBranch |  |
| 1. Die Datei example.txt mit neuem, beliebigen Text füllen und die Änderungen commiten |  |
| 1. Zurück in den Branch master wechseln |  |
| 1. Die Änderungen des Branches *tutorialBranch* in *master* zusammenführen:   git merge tutorialBranch  Es entsteht ein merge-Konflikt. |  |
| 1. Die Datei *example.txt* mit einem Texteditor oder merge-Tool so bearbeiten, dass alle Änderungen behalten werden. |  |
| 1. Die gefixte Datei erneut stagen und commiten. In dem sich öffnenden Texteditor die Commit Nachricht anpassen und beschreiben, wie der merge-Konflikt gelöst wurde. |  |

Teil 7: Installation überprüfen

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

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.

|  |  |
| --- | --- |
| **Explanation** | **Screenshot** |
| 1. Öffnen des „Perspektive öffnen“ Dialogs |  |
| 1. Auswahl von „Git“ |  |
| 1. Im Browser das Repository unter dem Link   <https://github.com/Matthias-Str/iis_thema_1/>  aufrufen. Mit dem Button Fork eine Kopie des Repositorys anlegen. |  |
| 1. Die Repository URL kopieren |  |
| 1. Zurück in Eclipse „Clone a Git repository“ auswählen |  |
| 1. In dem Dialog die URI einfügen und auf finish klicken. Das Repository wird anschließend heruntergeladen. |  |
| 1. Wechseln in die Ansicht „Java“ |  |
| 1. Im Dropdownmenü „File“ → „Import“ auswählen. |  |
| 1. Nach git suchen, „Projects from Git“ markieren und auf Next klicken |  |
| 1. In nächsten Schritt „Existing local repository“ wählen und auf Next klicken. |  |
| 1. Das eben heruntergeladene Repository auswählen. |  |
| 1. Die automatische Auswahl benutzen. |  |
| 1. Eclipse zeigt die Liste der gefundenen Projekte, alle auswählen und den Dialog Finishen. |  |
| 1. Das Projekt wird im Package Explorer angezeigt. |  |
| 1. Wechseln zurück in die Ansicht Git. Rechtsklick auf das Repository und „Switch To -> New Branch“ auswählen. |  |
| 1. Den Namen TutorialPersistenz eingeben |  |
| 1. 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** |
| 1. MySQL-Service starten:   sudo service mysql start  **Passwort**: lars+glas |  |
| 1. MySQL -Konsole starten:   sudo mysql -u root -p  **Passwort**: master42 |  |
| 1. Existierende Datenbanken anzeigen lassen:   show databases; |  |
| 1. Neues Schema für Tutorial erstellen:   create database shop; |  |
| 1. Erneut alle existierenden DB anzeigen lassen:   show databases;  Jetzt sollte die DB shop auch erscheinen.  **Hinweis**:  Tabellen müssen nicht händisch angelegt werden, da diese vom JPA-Projekt automatisch entsprechend der angelegten Entities erstellt werden. |  |
| 1. Die erstellte Datenbank muss nun dem WildFly-Server als DataSource hinzugefügt werden. Dafür im Eclipse-Projekt:   *Rechtsklick auf den WildFly-Server > Add and Remove…* |  |
| 1. 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*). |  |
| 1. Klappt man nun den WildFly-Server auf, sollte dort das Projekt aufgelistet werden. |  |
| 1. Nun den Server starten und im Browser die Adresse localhost:9990 aufrufen. Das Starten dauert, und erst dann kann man die Adresse aufrufen. |  |
| 1. Anmelden mit   **User:** admin  **Passwort**: admin |  |
| 1. Im Reiter *Configuration* zu folgendem Pfad navigieren:   *Subsystems > Datasources > Non-XA*  Anschließend den blauen Button *Add* klicken. |  |
| 1. MySQL Datasource auswählen und mit *Next* weiter navigieren. |  |
| 1. In Schritt 1/3 die Felder folgendermaßen befüllen:   **Name**: MySqlShopdbDS  **JNDI Name**: java:jboss/datasources/Shop  Weiter mit *Next*. |  |
| 1. In Schritt 2/3 im Reiter *DetectedDriver* den bereits hinterlegten mysql-Driver wählen und mit *Next* weiter navigieren. |  |
| 1. In Schritt 3/3 folgende Attribute eingeben:   **Connection URL:** jdbc:mysql://localhost:3306/shop  **Username**: root  **Passwort**: master42  Mit *Next* weiter navigieren. |  |
| 1. Eingegebene Daten im Summary überprüfen und mit *Finish* beenden. |  |
| 1. Verbindung testen, indem im Dropdown neben der eben angelegten DataSource *Test Connection* ausgewählt wird. |  |
| 1. 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 Lehrezeichen, 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.servelet 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** |
| 1. Mit der rechten Maustaste auf das Paket thema1.tutorial.jdbc.servlet klicken und im Kontexmenü unter *New > Servlet* ein neues Servelt erstellen |  |
| 1. Im sich öffnenden Fenster den Klassennamen ConnectionTest eingeben und mit *Finish* beenden.   Das neu erstellte Servlet wird geöffnet. Es soll einen Konstruktor sowie die Klassen doGet() und doPost() enthalten. |  |
| 1. Im Code zunächst die nebenstehenden Zeilen nach dem Attribut serialVersionUID einfügen. | **Hintergrund:**  Diese DataSource haben wir zuvor in unserem WildFly-Server als Data Source hinterlegt. Durch die Annotation @Resource und das Attribut lookup, sorgt der Web-Container beim Aufruf der DataSource dafür, diese zu injizieren und nutzt dabei den JNDI (Java Naming und Directory Interface). Dabei ist die DataSource eine Verbindung zur hinterlegten Datenbank, wobei es möglich ist, mehrere Connections gleichzeitig aufzubauen. |
| 1. Sollte DataSource wegen fehlender Imports rot unterringelt 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. |  |
| 1. 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();  } |
| 1. Das Servlet ist nun fertig und kann gestartet werden. Dafür zunächst das Projekt neu bauen:   *Rechtsklick auf das Projekt > Build Project* |  |
| 1. Nun das Projekt auf dem WildFly-Server neu publizieren:   *Rechtsklick auf das Projekt im Server > Full Publish*  Danach muss neben TutorialThema1 [Started, Synchronized] stehen. |  |
| 1. Nun kann das Servlet ausgeführt werden. Dafür:   *Rechtsklick im Fenster des Servlet-Codes > Run as > Run on Server* |  |
| 1. 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. |  |
| 1. 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** |
| 1. Eine neue Entity-Klasse erstellen:   *Rechtsklick auf das Package* thema1.tutorial.jpa.entities *> new > JPA Entity* |  |
| 1. Im neuen Fenster den Klassennamen Customer eintragen und mit *Next* weiter navigieren. |  |
| 1. Nun können die Attribute des Entitys angelegt werden. Dazu den Button *Add…* klicken. |  |
| 1. Im kleinen Fenster die Attribute des Fields setzten:   **Type:** int  **Name:** id  Mit *OK* bestätigen. |  |
| 1. 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 | |
| 1. 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. |  |
| 1. 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. |  |
| 1. Die Funktion setId() muss gelöscht werden, da wir später die id automatisch generieren lassen wollen. |  |
| 1. 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…* |  |
| 1. 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. |  |
| 1. 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/()…* |  |
| 1. 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. |  |
| 1. 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()…* |  |
| 1. 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. | |
| 1. 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*). |  |
| 1. Die Attribute firstname, lastname und email sollen keine Nullwerte erlauben, was durch die Annotation @Column definiert wird.   Dazu vor die drei Attribute entsprechend des nebenstehenden Codes die Annotation hinzufügen. |  |
| 1. Die Entity-Beziehungen werden über Annotation hinzugefügt.   Dafür vor dem Attribut adresse die Annotation @OneToOne und vor das Attribut orders die Annotation @OneToMany hinzufügen (entsprechend nebenstehendem Code). | **Hintergrund:**  Über den CascadeType wird festgelegt, dass die Liste mit Orders automatisch persisitert 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. |
| In der Datei persistence.xml ist über ein Attribut festgelegt, dass die Entitäten beim Start der Applikationin der Datenbank automatisch erstellt werden und beim Beenden gelöscht werden. Im Folgenden prüfen wir, ob diese Eigenschaft erfüllt wird, indem wir zunächst den aktuellsten Stand auf den Server publizieren und anschließend in der Datenbank die Existenz der Tabellen nachvollziehen. | |
| 1. Das Projekt neu bauen:   Datei speichern, dann  *Rechtsklick auf das Projekt > Build Project* |  |
| 1. Den neuen Stand des Projekts auf den Server publizieren:   *Rechtsklick auf das Projekt im Server-Fenster > Full Publish*  **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 create-drop (in der Datei persistence.xml gesetzt) versucht wird, beim Starten alle Tabellen zu löschen, wobei in der Datenbank keinerlei Tabellen enthalten sind. | 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 |
| 1. Im Terminal, indem bereits vorher die Tabellen der Datenbank geprüft wurden, alle Tabellen anzeigen lassen:   In der Datenbank in das Shema 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 @ManyToOne Annotation haben, eine Join-Tabelle erstellt, wie Customer\_Order und Order\_Item. |  |

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 Enities zu erstellen, zu bearbeiten oder zu löschen.

|  |  |
| --- | --- |
| **Explanation** | **Screenshot** |
| 1. Erstellen einer neuen Service-Klasse:   *Rechtsklick auf das Paket thema1.tutorial.jpa.services > New > Other* |  |
| 1. Im Dialog nach bean suchen und im Ordner **EJB** „Session Bean“ auswählen.   Weiter mit *Next*. |  |
| 1. 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. |  |
| 1. 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*) |  |
| 1. 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.  Nebenstehenden Code unterhalb des Konstruktors hinzufügen. Und abschließend Imports reorganisieren (*Strg+Shift+O*). Für die Liste wieder java.util.list wählen. | **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 zurückgegeben. Über getSingleResult() kann nur das erste Element zurückgegeben werden. |
| 1. Die erstellten Funktionen müssen nun auch dem Interface hinzugefügt werden.   Dazu in der Datei CustomerServiceLocal die zwei Zeilen der Funktionsdeklarationen entsprechend nebenstehendem Code hinzufügen.  Danach die Imports erneuern (*Strg+Shift+O*) und für *List* wieder java.util.List auswählen. |  |

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** |
| 1. Neues Paket für Servlets der Entity Customer erstellen.   *Rechtsklick auf src > New > Package* |  |
| 1. Namen des Packages eingeben:   thema1.tutorial.jpa.servlets.customer  Mit *Finish* beenden |  |
| 1. Erstellen eines neuen Servlets:   *Rechtsklick auf das Paket thema1.tutorial.jpa.servlets.customer> New > Other* |  |
| 1. 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. |  |
| 1. 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.  Somit können wir alle in dem Interface angebotenen Methoden nutzen und müssen im Servlet keine JPA-Methoden anwenden. |  |
| 1. Nebenstehenden Code in die doGet()-Methode kopieren und die Imports neu organisieren.   Für die Aktualisierung der Imports folgendes importieren:  java.sql.date  java.util.list  Datei speichern!  **Hintergrund:**  Zunächst wird ein neues Objekt Customer erstellt. Diesem wird eine Liste mit Bestellungen hinzugefügt sowie eine Adresse. Anschließend wird das Objekt persistiert. Am Ende werden alle Ergebnisse der Tabelle Customer gelesen, um zu prüfen, ob der Eintrag hinzugefügt wurde. | 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());  }); |
| 1. Nun ist das Servlet fertig und kann ausgeführt werden. Dazu muss zunächst das Projekt neu gebaut werden.   *Rechtsklick auf den Projektordner > Build Project* |  |
| 1. Nun das Projekt neu auf den Server publizieren:   *Rechtsklick auf das Projekt im WildFly-Server > Full Publish* |  |
| 1. Nun kann das Servlet gestartet werden:   *Rechtsklick in der Servlet-Datei > Run as > Run on Server* |  |
| 1. Die Ausgabe sollte der nebenstehenden entsprechen. |  |
| 1. Das Ergebnis kann auch in der Datenbank direkt überprüft werden:   select \* from Customer;  select \* from Adresse;  select \* from Orders; |  |

TEIL 14: Unit Tests

Teil 15: Änderungen in Repository laden

Teil 16: Projekt mit einer neuen Dependency (Hier LOG4J)

|  |  |
| --- | --- |
| **Explanation** | **Screenshot** |
| 1. Rechtsklick in den Project Explorer   ⭢ New  ⭢ Other… |  |
| 1. Auswahl im Wizzard: Maven Project   ⭢ Next |  |
| 1. Im nächten Schritt „Create a simple project (skip archetype selection)“ anwählen   ⭢ Next |  |
| 1. 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 |  |
| 1. 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 |  |
| 1. Im linken Dependencies Abschnitt auf „Add“ klicken, es öffnet sich folgender Dialog |  |
| 1. In dem Suchfenster „log4j“ eingeben, nicht mit Enter bestätigen sondern einfach kurz warten |  |
| 1. log4j mit einfachem klick auswählen und nachdem die Text Felder automatisch befüllt wurden auf ok drücken |  |
| 1. 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 |  |
| 1. Nun rechts unter Dependency Management auf Add drücken, es öffnet sich wieder ein ähnlicher Dialog |  |
| 1. Wie oben log4j über die Suche auswählen   ⭢ ok |  |
| 1. In den Package Explorer wechseln und Rechtsklick auf src/main/java |  |
| 1. ⭢ New   ⭢ Class |  |
| 1. Als Namen LoggingProgram eingeben   ⭢ Finish |  |
| 1. Code mit dem Inhalt von LoggingProgram.java aus Moodle ersetzen   **Anmerkung**: Nicht in Panik geraten wenn IntelliJ die Imports noch nicht kennt. |  |
| 1. Strg + Shift + S drücken um alle Änderungen zu speichern!   Rechtsklick auf das Projekt ⭢ Run As ⭢ Maven install |  |
| 1. Im Konsolenfenster sollte nun BUILD SUCCESS erscheinen |  |
| 1. 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 |  |
| 1. Ausgabe sollte folgendermaßen aussehen: |  |

Teil 17: Maven automatisiertes deployen

|  |  |
| --- | --- |
| **Explanation** | **Screenshot** |
| 1. Im Projekt orderhandling-wildfly-process die pom.xml öffnen und in die XML Ansicht wechseln |  |
| 1. Unter dem Tag <build> den <plugins> tag suchen und dort den Inhalt der Datei deploy\_plugin.txt einfügen |  |
| 1. Überprüfen ob die Syntax korrekt ist und der Inhalt der Datei (beginnend und endend mit dem <plugin> tag) richtig im eingefügt wurde.   **Anmerkung**: Da in unserem Projekt alles auf einer Maschiene läuft sind keine Uploads nötig. Für das simple Kopieren gibt es mehrere Möglichkeiten und verschiedene Plugins. In diesem StackOverflow Post sind einige davon aufgelistet:  <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 |  |
| 1. 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 |  |
| 1. 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** |
| 1. 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). |  |
| 1. **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. |  |
| 1. Das Ergebnis kann auch in der Datenbank überprüft werden.   Im Terminal:  mysql -u root -p  Passwort = master42 |  |
| 1. Anschließend die Datenbank shop auswählen:   use shop; |  |
| 1. Beim selektieren aller Zeilen in der Tabelle Adresse sollten zwei Einträge angezeigt werden:   select \* from Adresse; |  |
| 1. **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. |  |
| 1. Auch dieses Ergebnis kann in der Datenbank kontrolliert werden, indem die Schritte 3 und 4 ausgeführt wurden und anschließend Schritt 5 erneut ausgeführt wird. |  |
| 1. **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. |  |
| 1. Auch dieses Ergebnis kann in der Datenbank kontrolliert werden, indem die Schritte 3 und 4 ausgeführt wurden und anschließend Schritt 5 erneut ausgeführt wird. |  |
| 1. **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. |  |

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 *File > New > Dynamic Web Project* wählen. |  |
| 1. Anschließend den Projektnamn eingeben und prüfen, dass alle Einstellungen den nebenstehenden entsprechen.   Mit *Finish* beenden. |  |
| 1. 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…* |  |
| 1. 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. |  |
| 1. Hier den Type der JPA Implementation auf *Library Provided by Target Runtime* setzen.   Mit *Finish* beenden. |  |
| 1. Nun müssen in der Datei persistence.xml einige Änderunge vorgenommen werden. Dazu die Datei öffnen.   Hintergrund:  In dieser Datei werden alle Einstellungen für eine Pesristenzeinheit gespeichert. Dazu gehört unter Anderem der Provider, sowie die Datenbank. |  |
| 1. 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. |  |
| 1. Im Reiter Connection wir die JTA DataSource hinterlegt.   java:jboss/datasource/Shop  Der genaue Name lässt sich aus der für WildFly konfigurierten DataSource entnehmen. Dabei JNDI nutzen. |  |
| 1. Nun zum Reiter **Hibernate** wechseln, um die Datenbank genauer zu spezifizieren.   Im DropDown *Database dialect* wird MySQL ausgewählt. |  |
| 1. Als Default Schema den Namen des zu nutzenden Schemas angeben.   Im Tutorial: shop  Änderungen speichern. |  |
| 1. 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! |  |