HOCHSCHULE HEIDELBERG
Intelligence in Learning

JPA WITH HIBERNATE & ECLIPSELINK

DEMO

Software Development Practice

SRH University – Heidelberg
Prof. Dr Ajinkya Prabhune

1. INTRODUCTION

This is a technical demo which will guide you through step by step how you can create sample application with JPA with Hibernate and Eclipselink. In this demo we will be working with student details for understanding. This will be kick start project for proper understanding.

1.1 Pre-requisite

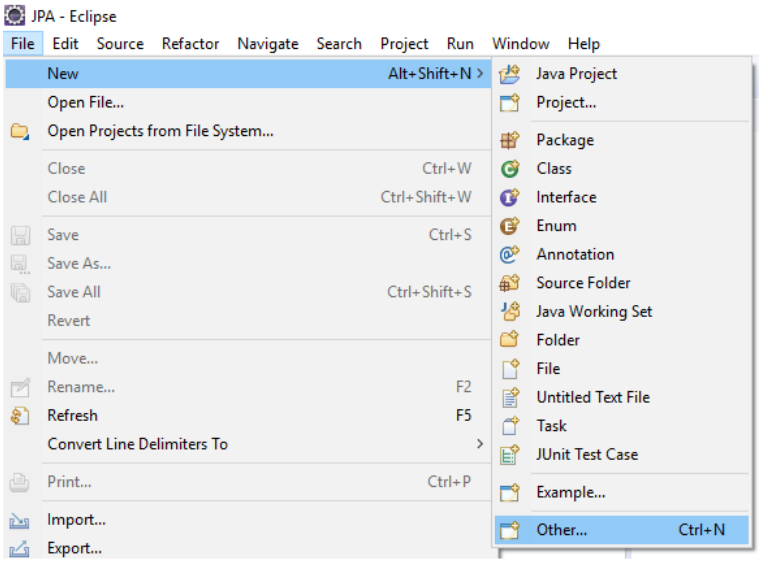
- Eclipse
- MySQL

1.2 Important Links

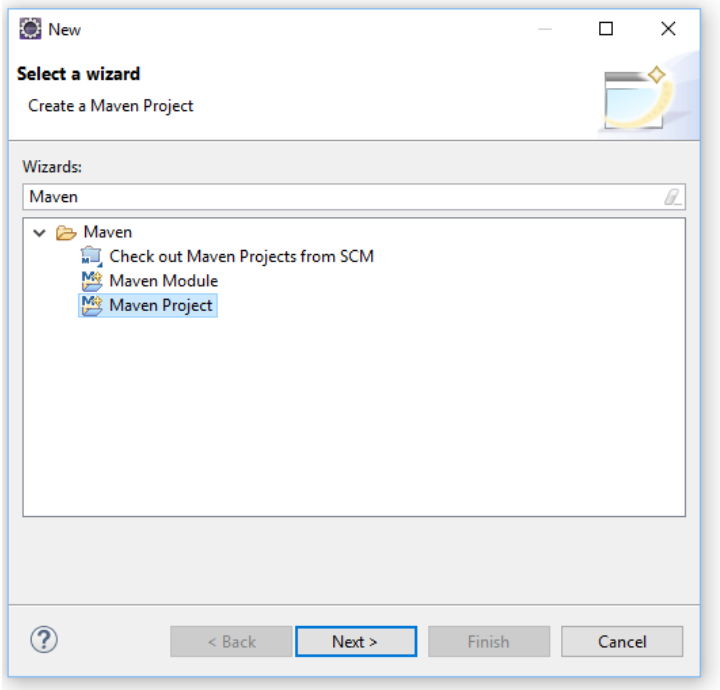
- Eclipse Download – https://www.eclipse.org/downloads/
- MySQL Download - https://dev.mysql.com/downloads/installer/
- GitHub Source Code - https://github.com/SRH-SDP/Exercises-Code

2. Creating Project

2.1 Selection of project

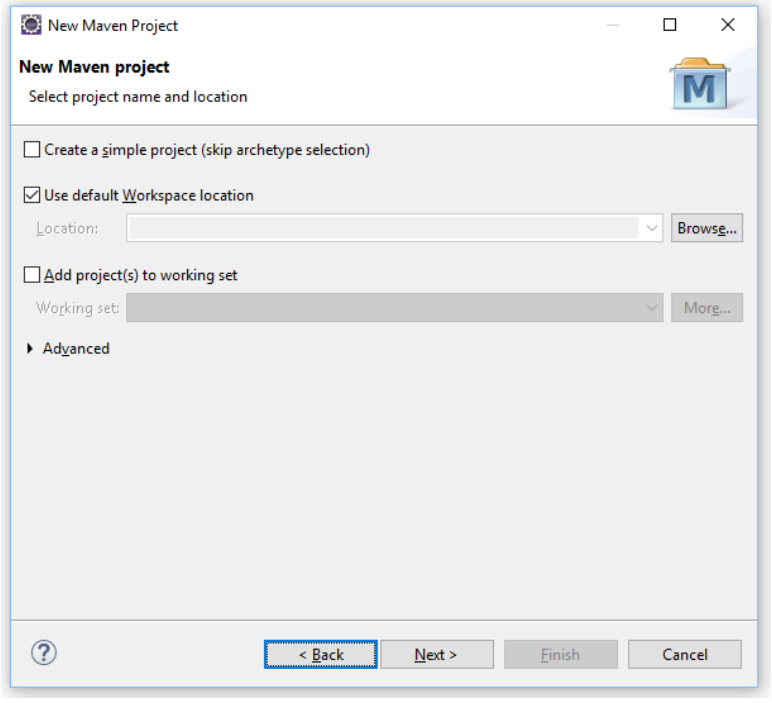


2.2 Selecting Maven Project

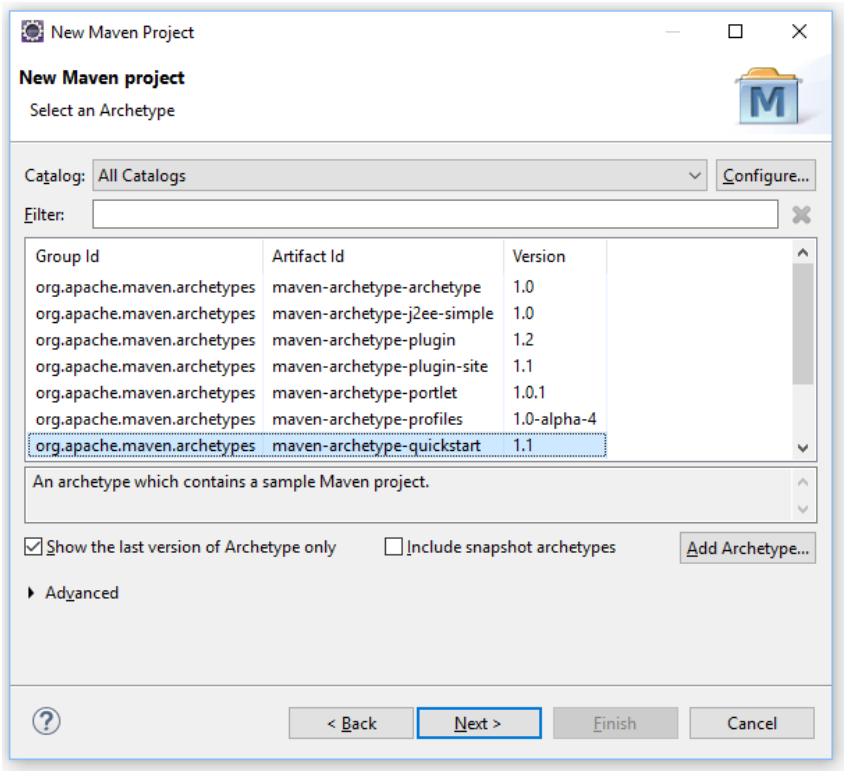


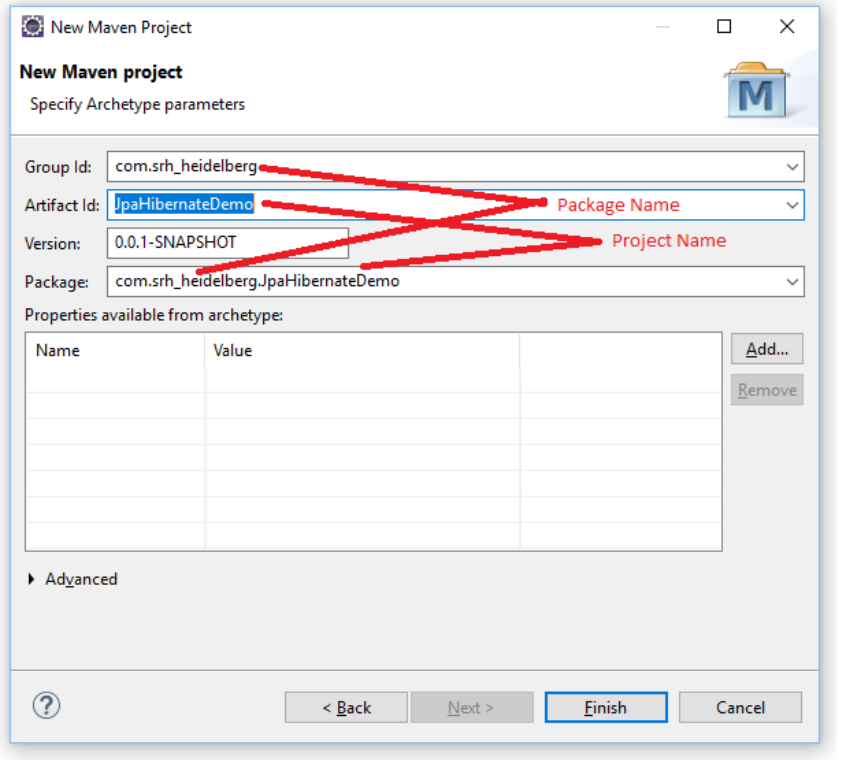
2.3 Maven setup

Keep this page with default selection



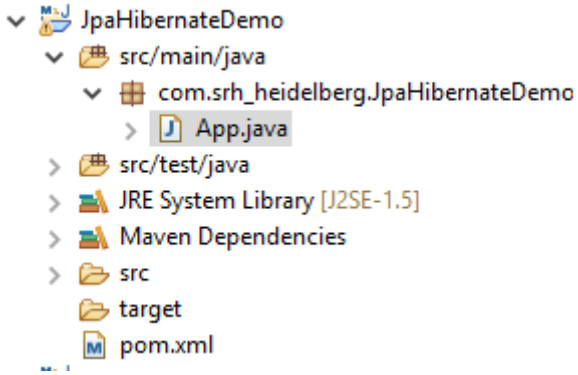
When you click next on this page you will get screen where you will need to select archetype of the project you will need to select “maven-archetype-quickstart” you can refer screen shot below.





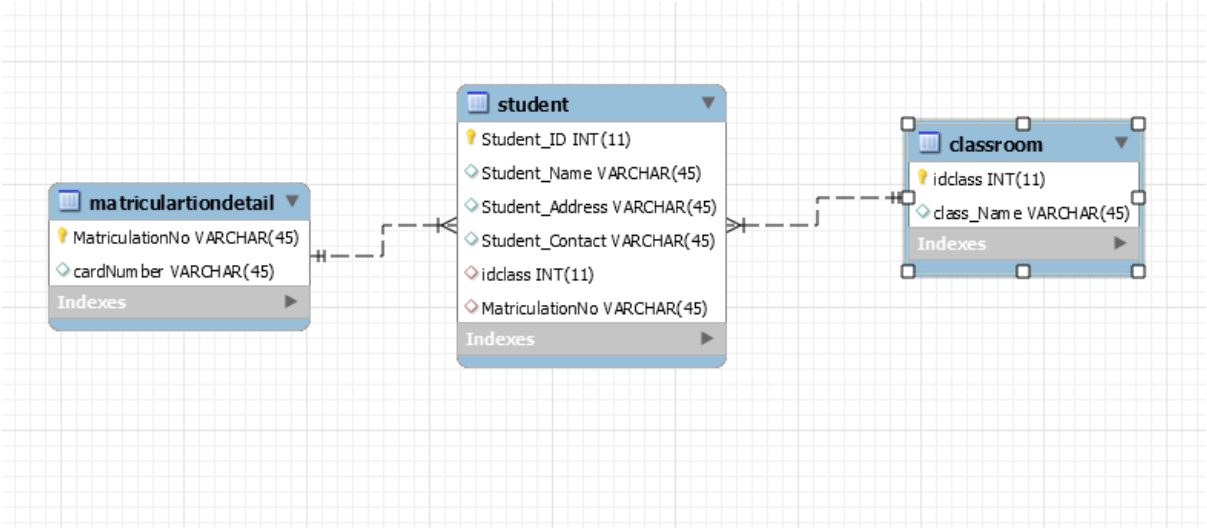
2.4 Project files

On successful creation of project, you will see below shown default project structure in Package Explorer of Eclipse IDE.



3. Database Setup

In this demo we are using MySQL as database server. Let us consider sample table of “Students” with minimal inputs which will be using schema “JPA_DB”



3.1 Creating Database

You will need to create schema with schema name “JPA_DB” for this example. Below given is query for the same.

```
CREATE SCHEMA jpa_db ;
```

After creating schema

```
CREATE TABLE `student` (
  `Student_ID` int(11) NOT NULL AUTO_INCREMENT,
  `Student_Name` varchar(45) DEFAULT NULL,
  `Student_Address` varchar(45) DEFAULT NULL,
  `Student_Contact` varchar(45) DEFAULT NULL,
  `idclass` int(11) DEFAULT NULL,
  `MatriculationNo` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`Student_ID`),
  KEY `class_Fk_idx` (`idclass`),
  KEY `mat_Fk_idx` (`MatriculationNo`),
  CONSTRAINT `class_Fk` FOREIGN KEY (`idclass`) REFERENCES `classroom` (`idclass`) ON DELETE NO ACTION
ON UPDATE NO ACTION,
  CONSTRAINT `mat_Fk` FOREIGN KEY (`MatriculationNo`) REFERENCES `matriculartiondetail`
(`MatriculationNo`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;

CREATE TABLE `classroom` (
  `idclass` int(11) NOT NULL AUTO_INCREMENT,
  `class_Name` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`idclass`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `matriculartiondetail` (
  `MatriculationNo` varchar(45) NOT NULL,
  `cardNumber` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`MatriculationNo`),
  UNIQUE KEY `MatriculationNo_UNIQUE` (`MatriculationNo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

4. Maven Dependencies

Maven has its own repository for dependencies you can visit https://mvnrepository.com/ you can get all the dependencies which can be used in project. For this example, we will require two dependencies.

4.1 Add Hibernate Dependencies

You can search in maven repository for hibernate core and get the dependencies code and paste it in the pom.xml file or use the below given code.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.8.Final</version>
</dependency>
```

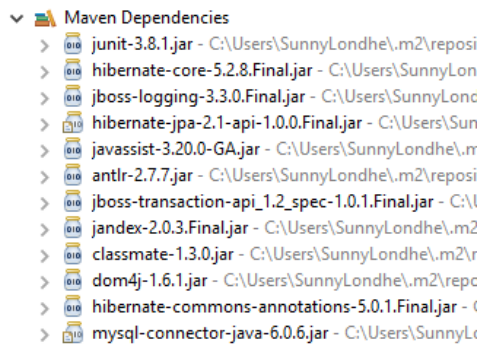
4.2 Add MySQL Dependencies

You can search in maven repository for MySQL and get the dependencies code and paste it in the pom.xml file or use the below given code. This is required to connect with database.

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>6.0.6</version>
</dependency>
```

4.2 Verify Dependencies

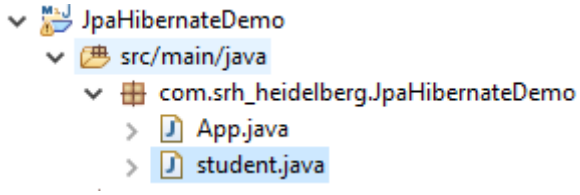
When you add dependencies code in pom.xml file of project maven will download all the required dependencies by itself so that you can use it in the project. Below given is the screenshot of the dependencies which can be viewed in package explorer.



5. Implementation

5.1 Create persistence class

Create simple class file in the package and name the class file same as the table name. As it will be representing the table in the database which will be called as Entity Model in this example. Below given is the screen shot of the same.



After creating class file we will need to add methods which will be same as the database columns. Along with that we will add getters and setters with tostring method for them. Below given is the same of the same.

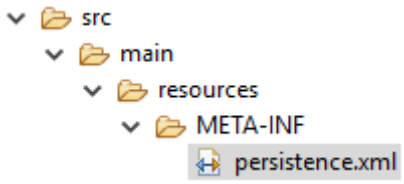
```
package com.srh_heidelberg.JpaHibernateDemo;

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

@Entity
public class student {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int Student_ID;
	private String Student_Name;
	private String Student_Address;
	private String Student_Contact;
	private int idclass;
	public int getIdclass() {
		return idclass;
	}
	public void setIdclass(int idclass) {
		this.idclass = idclass;
	}
```

```
	@ManyToOne(fetch=FetchType.LAZY)
	@JoinColumn(name = "idclass", insertable = false, updatable = false)
	private classroom classroom;

	public int getStudent_ID() {
		return Student_ID;
	}
	public void setStudent_ID(int student_ID) {
		Student_ID = student_ID;
	}
	public String getStudent_Name() {
		return Student_Name;
	}
	public void setStudent_Name(String student_Name) {
		Student_Name = student_Name;
	}
	public String getStudent_Address() {
		return Student_Address;
	}
	public void setStudent_Address(String student_Address) {
		Student_Address = student_Address;
	}
	public String getStudent_Contact() {
		return Student_Contact;
	}
	public void setStudent_Contact(String student_Contact) {
		Student_Contact = student_Contact;
	}
	@Override
	public String toString() {
		return "student [Student_ID=" + Student_ID + ", Student_Name=" +
Student_Name + ", Student_Address="
				+ Student_Address + ", Student_Contact=" +
Student_Contact + ", idClass=" + idclass + "]";
	}

}
```

Similarity create class for other tables.

5.2 Application and Database connection

You will need to create folder in “src\main” with names “resources” and within resources “META-INF” and inside that add persistence.xml file (Keep exact file names). The significance of this folder structure is that the persistence.xml file used for getting connection from the database. You will need to follow specific structure for providing details in the file for database connection exact code is given below. Alter the code as per your environment.



Contents of persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
     xmlns="http://xmlns.jcp.org/xml/ns/persistence"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                  http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="con">
        <properties>
           <property name="javax.persistence.jdbc.driver"
                     value="com.mysql.jdbc.Driver"/>
           <property name="javax.persistence.jdbc.url"
                     value="jdbc:mysql://localhost:3306/jpa_db"/>
           <property name="javax.persistence.jdbc.user" value="root"/>
           <property name="javax.persistence.jdbc.password" value="yourPassword"/>
        </properties>
    </persistence-unit>
</persistence>
```

5.3 CRUD Operations using JPA with Hibernate

You will need to add following sample code in “App.java” file. In this example we have performed Select, Insert, Update, Delete in one single code. Which is managed using one variable which is “CodeRunValue” you will need to change its values according to your requirements for example if you want to use Select option you will need to set value of CodeRunValue = 1, respectively for others.

```
package com.srh_heidelberg.JpaHibernateDemo;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class App {
	public static void main(String[] args) {
	 int CodeRunValue = 4; // 1 -> for select 2 -> for Insert 3 -> for
	                          Update 4 -> for Delete

	EntityManagerFactory emf = Persistence.createEntityManagerFactory("con");
	 EntityManager em = emf.createEntityManager(); // Entity Manager requires
	                                                  Entity Manager Factory
		if (CodeRunValue == 1) {
			// Selecting column with id 1
			student stuSelect = em.find(student.class, 1);
			System.out.println(stuSelect);
		} else if (CodeRunValue == 2) {
			// For Adding value in database
			student stuInsert = new student();
			stuInsert.setStudent_Name("Moris Abon");
			stuInsert.setStudent_Address("Mannheim");
			stuInsert.setStudent_Contact("+49 7777777777");
			stuInsert.setIdclass(1);
			em.getTransaction().begin(); // need to begin the
			                             transaction before persist
			em.persist(stuInsert);// persist will save value in app memory
			em.getTransaction().commit();// to save and reflect value into
			                              database.
			System.out.println(stuInsert);
		}else if(CodeRunValue == 3) {
			student stuUpdate = em.find(student.class, 3);
			em.getTransaction().begin(); // need to begin the transaction
			                              before persist
			stuUpdate.setStudent_Name("Jhon Mora");
			stuUpdate.setIdclass(2);
```

```
                em.getTransaction().commit();// to save and reflect value into
                                                database.
                System.out.println("Updated Successfully");
            }
            else if(CodeRunValue == 4) {
                student stuDelete = em.find(student.class, 4);
                em.getTransaction().begin(); // need to begin the transaction
                                                before persist
                em.remove(stuDelete); // persist will save value in app memory
                em.getTransaction().commit();// to save and reflect value into
                                                database.
                System.out.println("Deleted Successfully");
            }else {
                System.out.println("Invalid Operation Selected");
            }
        }
}
```

6. EclipseLink Demo

You will need to follow same steps which are mentioned in this document. You will need to skip point no 4 and only 5.2 completely below mentioned are the changes you need to do for eclipselink.

6.1 Introduction to Eclipselink with JPA

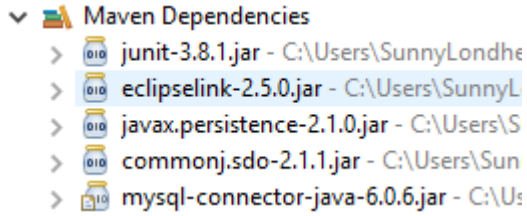
As you have already learnt about JPA with Hibernate you are now aware that JPA is a standard or even we can say that it's a specification for creating making application in ORM (Object Relational Mapping). And you also know that Eclipselink and Hibernate is ORM tool. Hence, we can use JPA specification to communicate with database with minimal changes in configuration of the application not in the main logic. Hence, we have kept same example and this time we will use eclipselink with maven.

6.2 Changes for using Eclipselink

- **Adding Dependencies of Eclipselink**
  In pom.xml remove dependency of hibernate and add Eclipselink dependency which is given below.

```
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>eclipselink</artifactId>
    <version>2.5.0</version>
</dependency>
```

- **Verify Dependencies**
  If you have added eclipselink dependency in proper way you will see maven dependencies structure as shown below.



- **Changes in persistence.xml**
  Eclipselink requires class definition in persistence.xml file and also you need to add property for eclipselink logging click here for more information about logging. Below are the necessary changes which is done for this demo.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
             xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
             http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="con">
            <class>com.srh_heidelberg.JpaEclipseLinkDemo.student</class>
            <class>com.srh_heidelberg.JpaEclipseLinkDemo.classroom</class>
        <properties>
            <property name="javax.persistence.jdbc.driver"
                      value="com.mysql.jdbc.Driver"/>
       <property name="javax.persistence.jdbc.url"
                 value="jdbc:mysql://localhost:3306/jpa_db"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.password" value="pwd2017*"/>
            <property name="eclipselink.logging.level" value="FINEST" />
        </properties>
 </persistence-unit>
</persistence>
```