# Exercise: Working with JPA and Hibernate

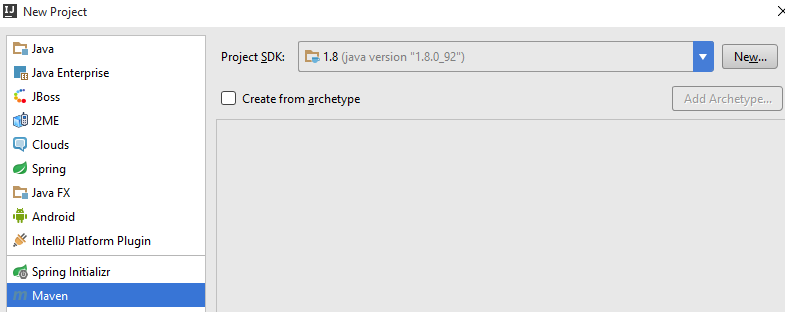
The purpose of this exercise is to make you **familiar** with the **JPA** (Java Persistence API). After completing it, you should be able to **write and execute queries that manipulate your MySQL database from Java** using JPA.

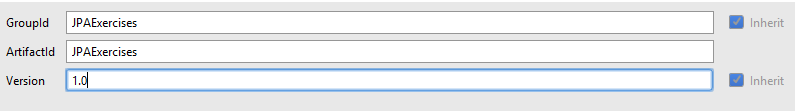
This lab is part of the [“Software Technologies” course @ SoftUni](https://softuni.bg/courses/software-technologies).

# Part I: Create a JPA Project

First, create a **Java** project and configure **JPA** with **Hibernate** implementation to access a **MySQL** database using **Maven** as a build system.

## Create Maven-Based Java Project







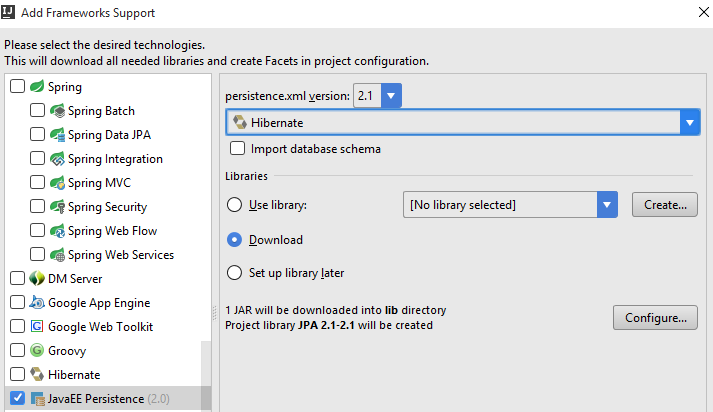
## Configure Maven

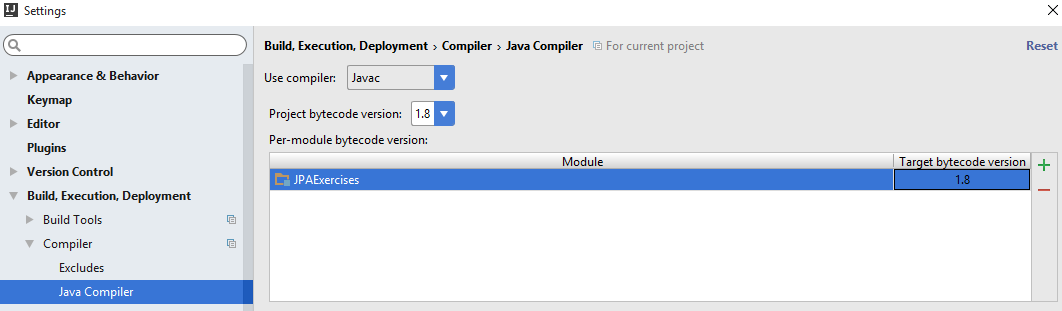
Add the following code in the Maven build file pom.xml:

|  |
| --- |
| pom.xml |
| <**build**>  <**plugins**>  <**plugin**>  <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-compiler-plugin</**artifactId**>  <**configuration**>  <**source**>1.8</**source**>  <**target**>1.8</**target**>  </**configuration**>  </**plugin**>  </**plugins**> </**build**>  <**dependencies**>  <**dependency**>  <**groupId**>org.hibernate</**groupId**>  <**artifactId**>hibernate-core</**artifactId**>  <**version**>5.2.1.Final</**version**>  </**dependency**>   <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>6.0.3</**version**>  </**dependency**> </**dependencies**> |

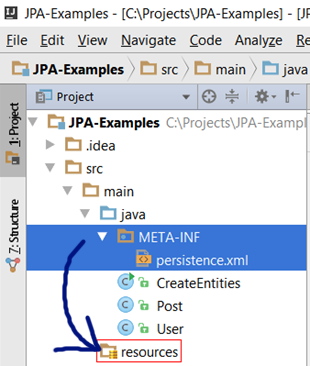
## Add JPA Support







Move the META-INF folder to the resources folder.



## Configure JPA

And replace your persistence.xml code with this one:

|  |
| --- |
| persistence.xml |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="2.1"**>  <**persistence-unit name="JPAExercises" transaction-type="RESOURCE\_LOCAL"**>  <**properties**>  <**property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"** />  <**property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/javablogdb"** />  <**property name="javax.persistence.jdbc.user" value="root"** />  <**property name="javax.persistence.jdbc.password" value=""** />  <**property name="hibernate.show\_sql" value="true"**/>  <**property name="hibernate.format\_sql" value="true"**/>  <**property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect"** />  <**property name="hbm2ddl.auto" value="update"**/>  </**properties**>  </**persistence-unit**> </**persistence**> |

# Part II: Define Entity Classes

## Import the MySQL Database

You are given **MySQL database script**. Import the script using **MySQL Workbench** / **HeidiSQL** / **phpMyAdmin** or any other tool that works for you.

[TODO: screenshot]

Now you need to create the classes for our entities. Here is the code for the classes.

## User Entity Class

|  |
| --- |
| User.java |
| **import** javax.persistence.\*; **import** java.util.HashSet; **import** java.util.Set;  @Entity @Table(name = **"users"**) **public class** User {   **private** Integer **id**;   **private** String **username**;   **private** String **fullname**;   **private** String **passwordhash**;   **private** Set<Post> **posts** = **new** HashSet<>(0);   **private** Set<Comment> **comments** = **new** HashSet<>(0);   @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  **public** Integer getId() {  **return id**;  }   **public void** setId(Integer id) {  **this**.**id** = id;  }   @Column(nullable = **false**)  **public** String getUsername() {  **return username**;  }   **public void** setUsername(String username) {  **this**.**username** = username;  }   **public** String getFullname() {  **return fullname**;  }   **public void** setFullname(String fullname) {  **this**.**fullname** = fullname;  }   @Column(nullable = **false**)  **public** String getPasswordHash() {  **return passwordhash**;  }   **public void** setPasswordHash(String passwordhash) {  **this**.**passwordhash** = passwordhash;  }   @OneToMany(mappedBy = **"author"**)  **public** Set<Post> getPosts() {  **return posts**;  }   **public void** setPosts(Set<Post> posts) {  **this**.**posts** = posts;  }   @OneToMany(mappedBy = **"author"**)  **public** Set<Comment> getComments() {  **return comments**;  };   **public void** setComments(Set<Comment> comments) {  **this**.**comments** = comments;  } } |

## Post Entity Class

|  |
| --- |
| Post.java |
| **import** javax.persistence.\*; **import** java.time.LocalDateTime; **import** java.util.HashSet; **import** java.util.Set;  @Entity @Table(name = **"posts"**) **public class** Post {   **private** Integer **id**;   **private** String **title**;   **private** String **content**;   **private** User **author**;   **private** LocalDateTime **date**;   **private** Set<Comment> **comments** = **new** HashSet<>(0);   **private** Set<Tag> **tags** = **new** HashSet<>(0);   @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  **public** Integer getId() {  **return id**;  }   **public void** setId(Integer id) {  **this**.**id** = id;  }   @Column(nullable = **false**)  **public** String getTitle() {  **return title**;  }   **public void** setTitle(String title) {  **this**.**title** = title;  }   @ManyToOne  **public** User getAuthor() {  **return author**;  }   **public void** setAuthor(User author) {  **this**.**author** = author;  }   @Column(nullable = **false**)  **public** String getContent() {  **return content**;  }   **public void** setContent(String content) {  **this**.**content** = content;  }   @Column(nullable = **false**)  **public** LocalDateTime getDate() {  **return date**;  }   **public void** setDate(LocalDateTime date) {  **this**.**date** = date;  }   @OneToMany(mappedBy = **"post"**)  **public** Set<Comment> getComments() {  **return comments**;  }   **public void** setComments(Set<Comment> comments) {  **this**.**comments** = comments;  }   @ManyToMany()  @JoinTable(  name=**"posts\_tags"**,  joinColumns={@JoinColumn(name=**"post\_id"**, referencedColumnName=**"id"**)},  inverseJoinColumns={@JoinColumn(name=**"tags\_id"**, referencedColumnName=**"id"**)})   **public** Set<Tag> getTags() {  **return tags**;  }   **public void** setTags(Set<Tag> tags) {  **this**.**tags** = tags;  } } |

## Comment Entity Class

|  |
| --- |
| Comment.java |
| **import** javax.persistence.\*; **import** java.time.LocalDateTime;  @Entity @Table(name = **"comments"**) **public class** Comment {   **private** Integer **id**;   **private** String **text**;   **private** Post **post**;   **private** User **author**;   **private** String **authorName**;   **private** LocalDateTime **date**;   @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  **public** Integer getId() {  **return id**;  }   **public void** setId(Integer id) {  **this**.**id** = id;  }   @Column(nullable = **false**)  **public** String getText() {  **return text**;  }   **public void** setText(String text) {  **this**.**text** = text;  }   @ManyToOne  @JoinColumn(name = **"post\_id"**)  **public** Post getPost() {  **return post**;  }   **public void** setPost(Post postId) {  **this**.**post** = postId;  }   @ManyToOne  @JoinColumn(name = **"author\_id"**)  **public** User getAuthor() {  **return author**;  }   **public void** setAuthor(User author) {  **this**.**author** = author;  }   @Column(name = **"author\_name"**)  **public** String getAuthorName() {  **return authorName**;  }   **public void** setAuthorName(String authorName) {  **this**.**authorName** = authorName;  }   @Column(nullable = **false**)  **public** LocalDateTime getDate() {  **return date**;  }   **public void** setDate(LocalDateTime date) {  **this**.**date** = date;  } } |

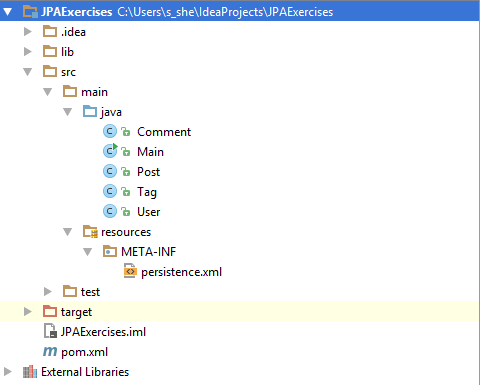
## Tag Entity Class

|  |
| --- |
| Tag.java |
| **import** javax.persistence.\*; **import** java.util.HashSet; **import** java.util.Set;  @Entity @Table(name = **"tags"**) **public class** Tag {  **private** Integer **id**;   **private** String **name**;   **private** Set<Post> **posts** = **new** HashSet<>(0);   @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  **public** Integer getId() {  **return id**;  }   **public void** setId(Integer id) {  **this**.**id** = id;  }   @Column(nullable = **false**)  **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  }   @ManyToMany()  @JoinTable(  name=**"posts\_tags"**,  joinColumns={@JoinColumn(name=**"tags\_id"**, referencedColumnName=**"id"**)},  inverseJoinColumns={@JoinColumn(name=**"post\_id"**, referencedColumnName=**"id"**)})  **public** Set<Post> getPosts() {  **return posts**;  }   **public void** setPosts(Set<Post> posts) {  **this**.**posts** = posts;  } } |

## Main Class

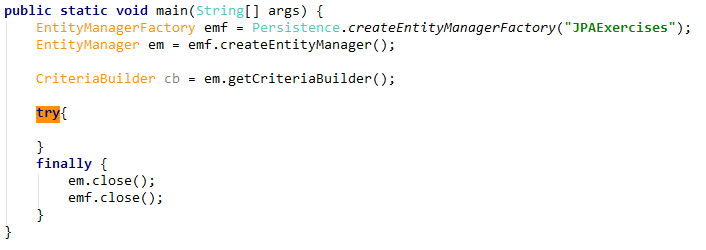
After creating all of the classes, create a Main.java class. You need to create your main() method there.

When you are done with this, you should have the following project structure:



# Part III: JPA Queries

We will use methods for this part of the exercise, but in order to do that, you need to go to your main() method and write the following code:

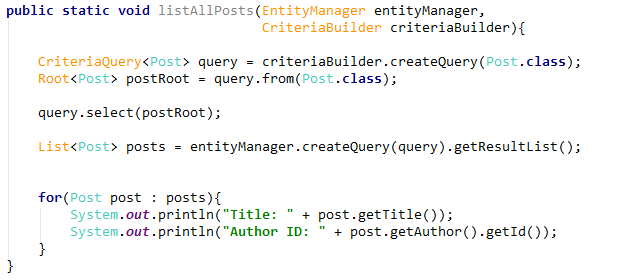


We will call our methods in the try block.

## Read Data

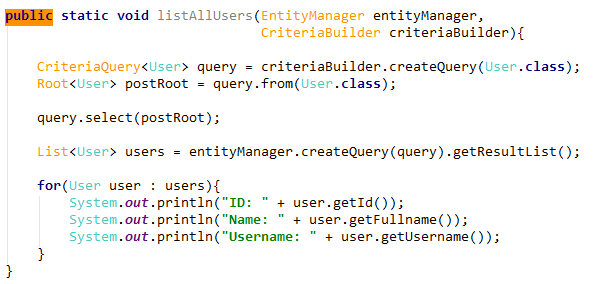
### List All Posts

Let’s list **all data rows** from the Posts table.   
Write the following code:



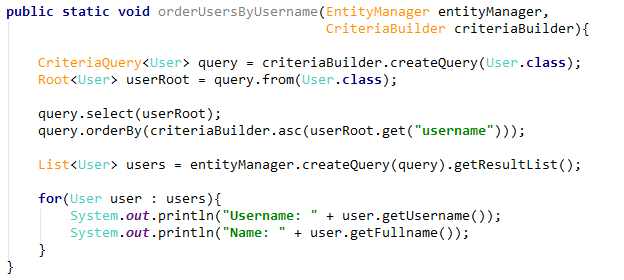
### List All Users

Let’s get **all users** now:



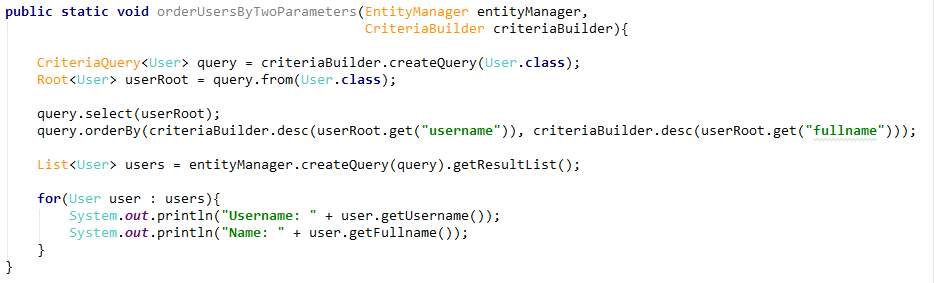
### Order Data

Let’s take all **usernames and full names** of our users. Now we will **order** the data in **ascending** order by **username**.



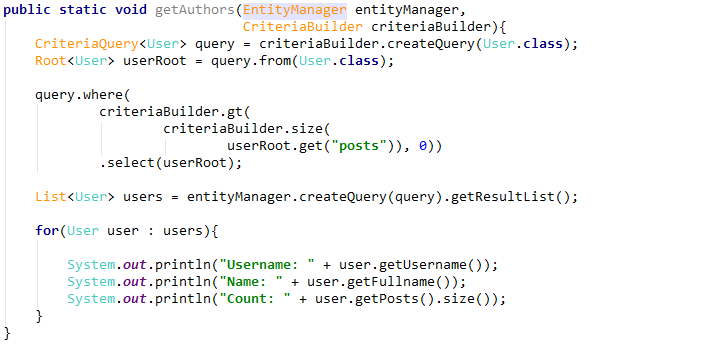
### Order by Two Columns

Let’s try to order by **2 parameters**, but this time we will do it in **descending** order:



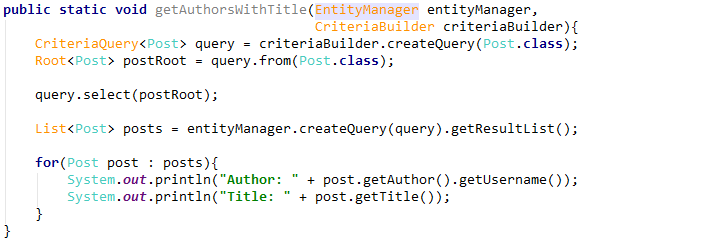
### Select Authors

**Not all** of our users have **created posts**. How can we get **only the users that have created a blog post**? Just like that:



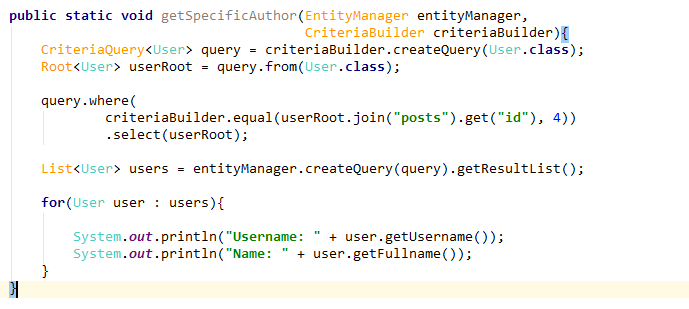
### Joins Authors with Titles

Something more interesting now – let’s get the usernames with the post names for each user. The query should look like this:



### Select Author of Specific Post

Let’s get the creator of post with id=4 now:



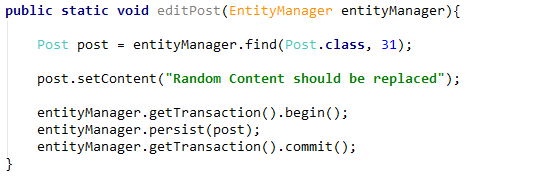
## Create Data

In order to **create** data, we need to **create object from the class** we desire to use. For this example, we are going to create a **new post**, which means that we need to create new **instance of** Post. Then we need to **save** our **database** with the new changes. Let’s try it!



## Update Data

If we want to **update** an object, first we need to **get it from the database.** Then we can **edit** its properties, and **finally save the database**. This is how it’s done:



## Delete Data

Now let’s delete the post that we’ve created earlier. We will do that, **using the ID** of the post. After getting the object, we should **delete everything that is linked** to it, from the **other tables**. In our case, we should delete the **tags and comments**. Then we can simple use Remove() to delete the post.

