Using JPA in a Java Desktop Application

This Document shows how to setup a project to connect to a database using the JPA implementation from EclipseLink.

Contents

[Requirements 2](#_Toc508956475)

[Eclipse Setup 2](#_Toc508956476)

[EclipseLink Download 2](#_Toc508956477)

[MySQL Java Connector Download 3](#_Toc508956478)

[Create User Library 3](#_Toc508956479)

[Project Setup 7](#_Toc508956480)

[Create JPA Project 7](#_Toc508956481)

[Create Connection 9](#_Toc508956482)

[Adjust persistence.xml 10](#_Toc508956483)

[Create Files 11](#_Toc508956484)

[Create Java Class from DB Table 11](#_Toc508956485)

[Use JPA 12](#_Toc508956486)

[Create small console interface 14](#_Toc508956487)

[Transaction example 15](#_Toc508956488)

[JDBC Example 15](#_Toc508956489)

[JPA Example 17](#_Toc508956490)

[Differences 18](#_Toc508956491)

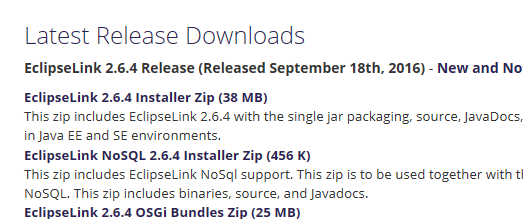
Last Changes: Luca Longo, 16.3.18 (09:39)

# Requirements

* Eclipse
* MySql Database (Do not delete the Schema “sakila”)
* Know how to create Classes and Packages in Eclipse

# Eclipse Setup

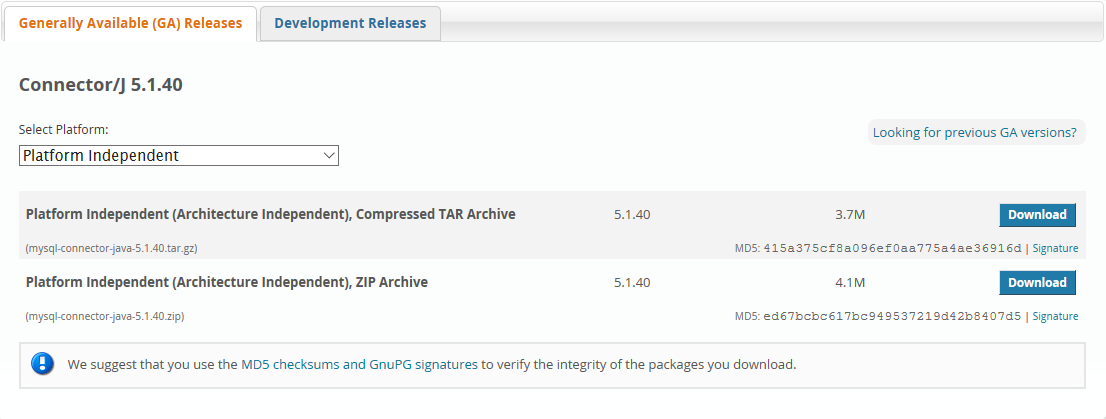
## EclipseLink Download

Download EclipseLink from [here](https://www.eclipse.org/eclipselink/downloads/): 

Screenshot from eclipse.org/eclipselink/downloads/

Now click the big download button.  
Once downloaded, extract the zip file to a convenient location (for example Desktop).

## MySQL Java Connector Download

If you have already downloaded a MySQL Java Connector (used by Glassfish), locate this file, otherwise download it from [here](https://dev.mysql.com/downloads/connector/j/): 

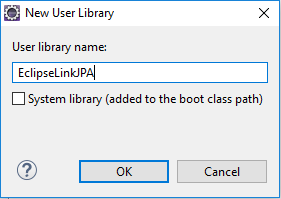
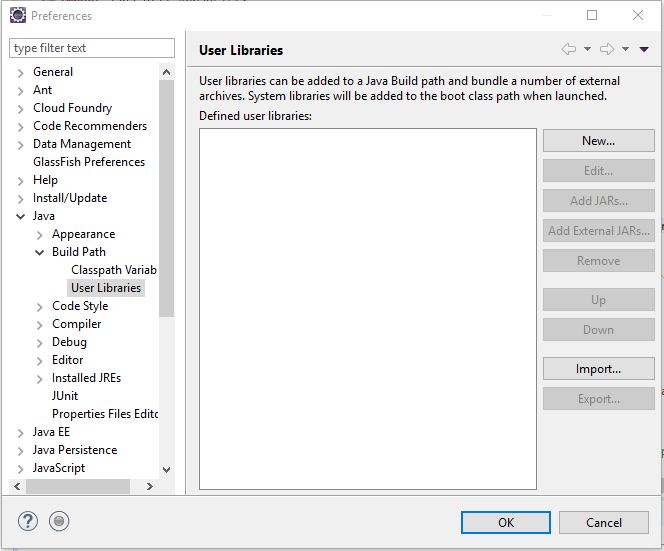
Screenshot from dev.mysql.com/downloads/connector/j/

Click “No thanks, just start my download.” to start the download without logging in.

## Create User Library

Now go to your Eclipse Folder (in my case C:\eclipse) and create a new Folder called “EclipseLinkJPA”. Copy following files into that folder:

* MySQL Java Connector
* From the extracted zip: eclipselink\jlib\eclipselink.jar
* From the extracted zip: eclipselink\jlib\jpa\\* (all files in the “jpa” folder)

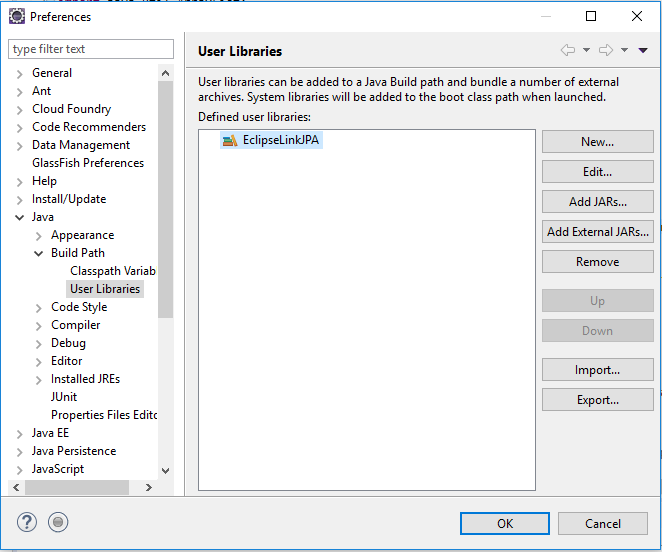
In Eclipse, go to *Window -> Preferences -> Java -> Build Path -> User Libraries* and create a new one: 

Screenshot of popup when clicking New…

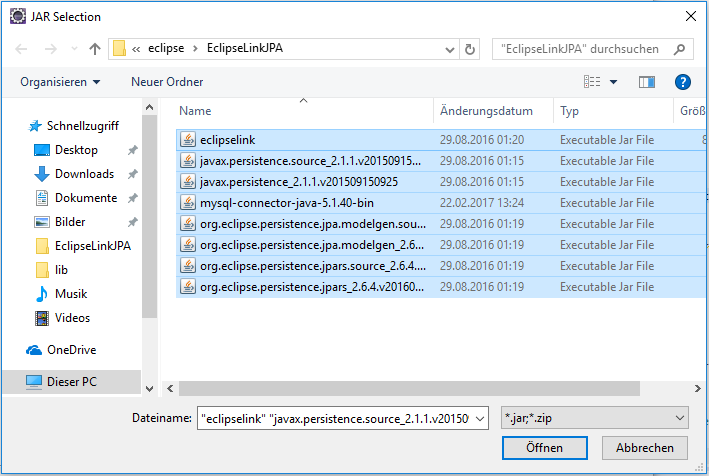
Screenshot of the preferences window in eclipse

Enter the text “EclipseLinkJPA” and click ok:

Now add the jars you copied before:



Screenshot of the preferences window in eclipse

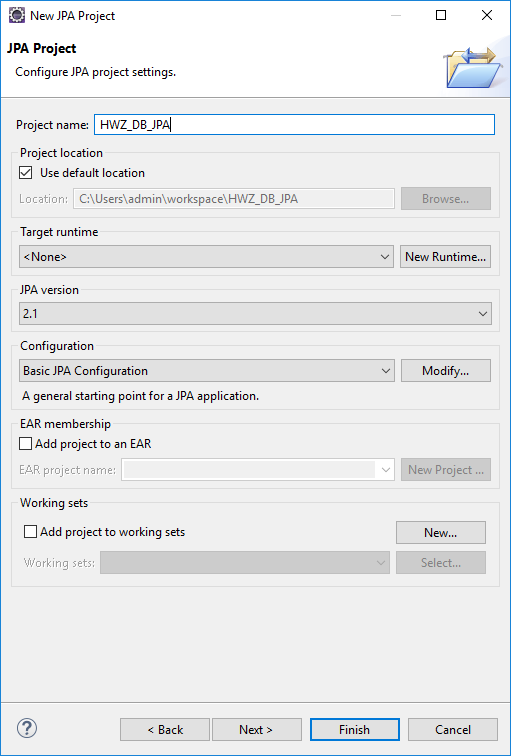


Screenshot of the file selector opened when clicking Add External Jars… navigated to folder created earlier

Navigate to the folder created earlier (EclipseLinkJPA in the eclipse installation), select all the files and click Open (in your system language, “Öffnen” in German) and then OK.

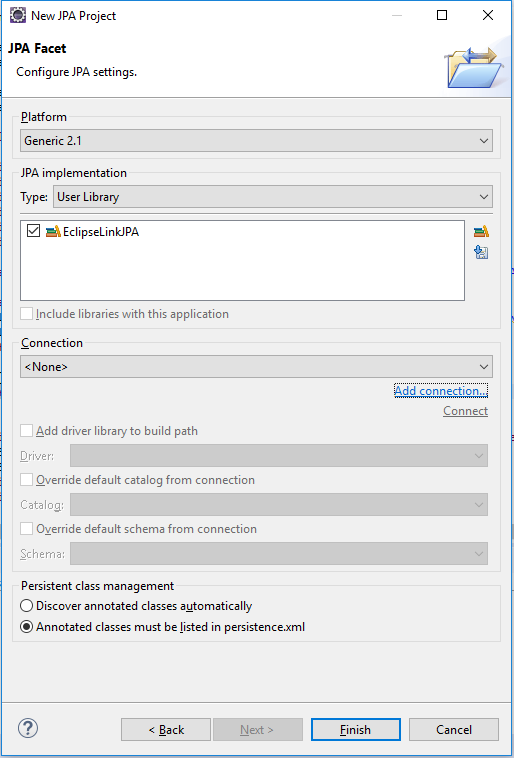
# Project Setup

## Create JPA Project

Create a new JPA Project *File -> New -> Other* (Shortcut: CTRL N) JPA Project. Call it “HWZ\_DB\_JPA” and leave everything as it is: 

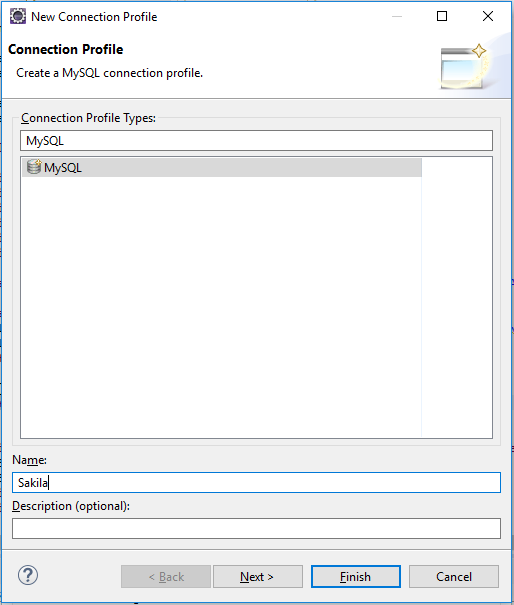
Screenshot of the first page in the JPA Project Wizard

Click Next **2 times**.

Select the newly created User Library as JPA Implementation and create a new connection: 

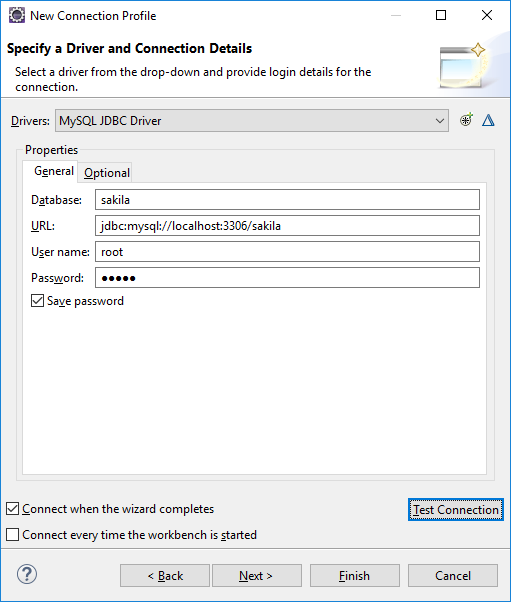
Screenshot of the last page in the JPA Project Wizard

## Create Connection

Select MySQL in the list and call the connection “Sakila”: 

Screenshot of the first page of the Connection Profile Wizard called when clicking Add connection…

In the next window, fill in the data as shown in the picture, where username and password are the credentials from your MySQL Database. Once finished, click Test Connection before you continue.



Screenshot of the second page of the Connection Profile Wizard

After clicking Finish, you will return to the JPA Project Wizard. The new connection should be selected automatically. Click Finish. (It will ask you to switch to JPA Perspective, but I recommend to stay in the current one)

## Adjust persistence.xml

In the new Porject, open the file persistence.xml which is locatet in *src -> META-INF.*

Replace its content with this: (user and password should again match your credentials in MySQL)

<?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=*"CountrySearch"*>

<properties>

<property name=*"eclipselink.ddl-generation"* value=*"drop-and-create-tables"*/>

<property name=*"eclipselink.logging.level"* value=*"OFF"*/>

<property name=*"eclipselink.jdbc.driver"* value=*"com.mysql.jdbc.Driver"*/>

<property name=*"eclipselink.jdbc.url"* value=*"jdbc:mysql://localhost:3306/sakila"*/>

<property name=*"eclipselink.jdbc.user"* value=*"root"*/>

<property name=*"eclipselink.jdbc.password"* value=*"admin"*/>

</properties>

</persistence-unit>

</persistence>

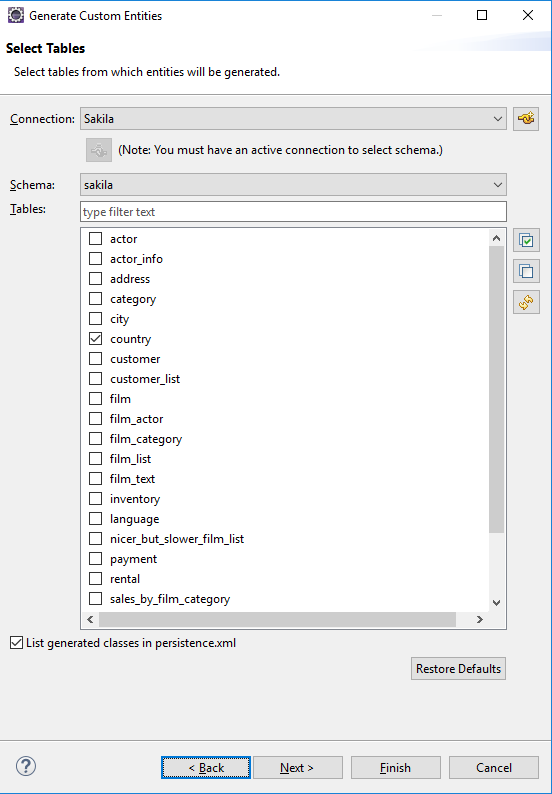
## Create Files

Next, create 2 new packages in src: “main” and “model”.  
The model package will contain code to make database requests, the main package will contain a small console application to test the model package.

Select the model package, *rightclick -> new -> other* (Shortcut: CTRL N) and select “JPA Entities from Tables”. Click Next.

### Create Java Class from DB Table

In the Connection Dropdown, ensure that Sakila is selected. Tick country from the Tables list. Click Finish.



Screenshot of the JPA Entities from Tables Wizard

### Use JPA

Also in the model package, create a new class called “CountryModel” and replace its code with the following:

**package** model;

**import** java.util.ArrayList;

**import** java.util.List;

**import** javax.persistence.EntityManager;

**import** javax.persistence.Persistence;

**public** **class** CountryModel {

**public** **static** **final** **int** ***BEGINS\_WITH\_OPTION*** = 1;

**public** **static** **final** **int** ***NOT\_BEGINS\_WITH\_OPTION*** = -1;

**public** **static** **final** **int** ***ENDS\_WITH\_OPTION*** = 2;

**public** **static** **final** **int** ***NOT\_ENDS\_WITH\_OPTION*** = -2;

**public** **static** **final** **int** ***CONTAINS\_OPTION*** = 3;

**public** **static** **final** **int** ***NOT\_CONTAINS\_OPTION*** = -3;

**private** EntityManager em = Persistence.*createEntityManagerFactory*("CountrySearch").createEntityManager();

**private** List<Country> searchCountry(String sqlCondition){

List<Integer> ids = em.createNativeQuery("Select country\_id from country where " + sqlCondition).getResultList();

List<Country> countries = **new** ArrayList<Country>();

**for**(**int** id : ids){

countries.add(getCountryById(id));

}

**return** countries;

}

**public** List<Country> searchCountry(String searchQuery, **int** searchOption, **boolean** caseSensitive){

String like = "LIKE";

String condition;

**if**(caseSensitive) like = like + " BINARY";

**if**(searchOption < 0){ //If the value is negative, it will be converted to its positive equal and NOT will be used

like = "NOT " + like;

searchOption \*= -1;

}

**switch** (searchOption) {

**case** ***BEGINS\_WITH\_OPTION***:

condition = "\"" + searchQuery + "%\"";

**break**;

**case** ***ENDS\_WITH\_OPTION***:

condition = "\"%" + searchQuery + "\"";

**break**;

**case** ***CONTAINS\_OPTION***:

condition = "\"%" + searchQuery + "%\"";

**break**;

**default**:

**throw** **new** IllegalArgumentException();

}

**return** searchCountry("country " + like + " " + condition);

}

**public** Country getCountryById(**int** id){

**return** em.find(Country.**class**, id);

}

}

### Create small console interface

In the package main, create a new class called “Main”. Replace it’s code with the following:

**package** main;

**import** java.util.Scanner;

**import** model.Country;

**import** model.CountryModel;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

CountryModel model = **new** CountryModel();

Scanner in = **new** Scanner(System.***in***);

System.***out***.println("Search country:");

String searchQuery = in.nextLine();

System.***out***.println("Select search option:");

System.***out***.println(" 1 : Begins with...");

System.***out***.println("-1 : Does not begin with...");

System.***out***.println(" 2 : Ends with...");

System.***out***.println("-2 : Does not end with...");

System.***out***.println(" 3 : Contains...");

System.***out***.println("-3 : Does not contain...");

**int** option = Integer.*parseInt*(in.nextLine());

System.***out***.println("Case sensitive? (Y/N):");

String caseSensitiveInput = in.nextLine().toUpperCase();

**boolean** caseSensitive = **false**;

**switch** (caseSensitiveInput) {

**case** "Y":

caseSensitive = **true**;

**break**;

**case** "N":

caseSensitive = **false**;

**break**;

**default**:

System.***out***.println("Invalid Input");

System.*exit*(0);

}

**for**(Country c : model.searchCountry(searchQuery, option, caseSensitive)){

System.***out***.println(c.getCountry());

}

in.close();

}

}

You can now test the application!

## Transaction example

### JDBC Example

To begin, create a new package called “jdbc\_transaction” in your project.  
Create an “Account” class and paste the following code in it:

**package jdbc\_transaction;**

**import java.sql.Connection;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.util.InputMismatchException;**

**import java.util.Scanner;**

**import exception.AmountTooHighException;**

**import utility.Utility;**

**public class Account{**

**private final int ID;**

**private int balance;**

**private Connection con;**

**private Scanner reader = new Scanner(System.in);**

**public Account(int ID, int balance, Connection con) {**

**this.ID = ID;**

**this.balance = balance;**

**this.con = con;**

**try {**

**PreparedStatement stmt = con.prepareStatement(**

**"insert into konto (ID, balance) select ?, ? where ? not in (select ID from konto)");**

**stmt.setInt(1, ID);**

**stmt.setInt(2, balance);**

**stmt.setInt(3, ID);**

**stmt.executeUpdate();**

**con.commit();**

**} catch (SQLException e) {**

**System.out.println(e);**

**}**

**}**

**public void transferMoneyTo(int destinationID, int amount) {**

**try {**

**int balance1 = getBalance() - amount;**

**updateKonto(balance1, this.ID);**

**Utility.waitTillEnter("Press Enter to continue");**

**int balance2 = getBalance(destinationID) + amount;**

**updateKonto(balance2, destinationID);**

**if (con.getAutoCommit() == false) {**

**con.commit();**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**try {**

**con.rollback();**

**} catch (SQLException e2) {**

**e2.printStackTrace();**

**}**

**}**

**}**

**public void updateKonto(int amountToOperateWith, int kontoID) {**

**try {**

**PreparedStatement stmt = con.prepareStatement("update konto set balance = ? where ID = ?");**

**stmt.setInt(1, amountToOperateWith);**

**stmt.setInt(2, kontoID);**

**stmt.executeUpdate();**

**} catch (SQLException e) {**

**System.out.println(e);**

**}**

**}**

**public int inputAmountToTake(String prompt, String errorMsg, String invalidMsg) {**

**int amount = 0;**

**boolean wrongInput = true;**

**while (wrongInput) {**

**try {**

**System.out.println(prompt);**

**amount = reader.nextInt();**

**} catch (InputMismatchException e) {**

**System.out.println(e + errorMsg);**

**reader.nextLine();**

**amount = getBalance() + 1;**

**}**

**try {**

**if (getBalance() > amount) {**

**wrongInput = false;**

**} else {**

**throw new AmountTooHighException(invalidMsg);**

**}**

**} catch (AmountTooHighException e) {**

**System.out.println(e + "\n");**

**}**

**}**

**return amount;**

**}**

**public int inputAmountToTake() {**

**return inputAmountToTake("Insert amount of money to be transferred from acc1 to acc2: ",**

**" Invalid Input! Try again", "Amount Input is too high! Try again");**

**}**

**public int getID() {**

**return ID;**

**}**

**public int getBalance() {**

**return getBalance(this.ID);**

**}**

**public int getBalance(int ID) {**

**int balance = 0;**

**try {**

**PreparedStatement stmt = con.prepareStatement("select balance as balance from konto where ID = ?");**

**stmt.setInt(1, ID);**

**ResultSet rs = stmt.executeQuery();**

**while (rs.next())**

**balance = rs.getInt("balance");**

**} catch (SQLException e) {**

**System.out.println(e);**

**}**

**return balance;**

**}**

**public void setBalance(int balance) {**

**try {**

**PreparedStatement stmt = con.prepareStatement("update konto set balance = ? where ID = ?");**

**stmt.setInt(1, balance);**

**stmt.setInt(2, this.ID);**

**stmt.executeUpdate();**

**con.commit();**

**} catch (SQLException e) {**

**System.out.println(e);**

**}**

**this.balance = balance;**

**}**

**public String toString() {**

**return "Konto " + this.getID() + ": " + this.getBalance();**

**}**

**}**

After that, create a new class called “TransactionApp” and use the following code:

**package** jdbc\_transaction;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**public** **class** TransactionApp {

**private** **static** Connection *connection*;

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, SQLException {

Class.*forName*("oracle.jdbc.driver.OracleDriver");

Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/sakila","root","admin");

Account acc1 = **new** Account(1, 1000, *connection*);

Account acc2 = **new** Account(2, 500, *connection*);

**boolean** loop = **true**;

**while** (loop) {

con.setAutoCommit(**false**);

System.***out***.println("Welcome!");

System.***out***.println(acc1);

System.***out***.println(acc2 + "\n");

**int** amount = acc1.inputAmountToTake();

acc1.transferMoneyTo(acc2.getID(), amount);

System.***out***.println("\nBalance: ");

System.***out***.println(acc1);

System.***out***.println(acc2 + "\n");

con.setAutoCommit(**true**);

System.***out***.println("Welcome!");

System.***out***.println(acc1);

System.***out***.println(acc2 + "\n");

amount = acc1.inputAmountToTake();

acc1.transferMoneyTo(acc2.getID(), amount);

System.***out***.println("\nBalance: ");

System.***out***.println(acc1);

System.***out***.println(acc2 + "\n");

}

}

}

You can now test the application!

### JPA Example

Before we begin, make sure that your persistence.xml file *(\src\****META-INF****\persistence.xml*) looks **exactly** like this (user and password should again match your credentials in MySQL):

<?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=*"JPA\_Connection"*>

**<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>**

**<class>transaction.Account</class>**

<properties>

<property name=*"javax.persistence.jdbc.driver"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"javax.persistence.jdbc.url"* value=*"jdbc:mysql://localhost:3306/sakila"* />

<property name=*"javax.persistence.jdbc.user"* value=*"root"* />

<property name=*"javax.persistence.jdbc.password"* value=*"admin"* />

</properties>

</persistence-unit>

</persistence>

Be sure to have a schema in your DB called “account”, containing the attributes “ID” and “balance”.

Here is the code used to create the schema:

**CREATE TABLE `account` (**

**`ID` int(3) NOT NULL AUTO\_INCREMENT,**

**`balance` int(11) NOT NULL,**

**PRIMARY KEY (`ID`)**

**)**

After making sure that the schema exists in the DB, create a new package called “transaction”.

Afterwards create a Java Entity class called “Account”.

Replace the existing code with the following:

**package transaction;**

**import javax.persistence.Column;**

**import javax.persistence.Entity;**

**import javax.persistence.GeneratedValue;**

**import javax.persistence.GenerationType;**

**import javax.persistence.Id;**

**import javax.persistence.Table;**

**@Entity**

**@Table(name = "account")**

**public class Account {**

**@Id**

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

**@Column(name = "ID")**

**private int ID;**

**@Column(name = "balance")**

**private int balance;**

**public int getID() {**

**return ID;**

**}**

**public void setID(int ID) {**

**this.ID = ID;**

**}**

**public int getBalance() {**

**return balance;**

**}**

**public void setBalance(int balance) {**

**this.balance = balance;**

**}**

**}**

Create a new Java class in the same package, call it “TransactionApp” and replace its code with the following:

**package transaction;**

**import javax.persistence.EntityManager;**

**import javax.persistence.EntityManagerFactory;**

**import javax.persistence.Persistence;**

**public class TransactionApp {**

**public static void main(String[] args) {**

**EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPA\_Connection");**

**EntityManager em = emf.createEntityManager();**

**Account acc1 = new Account();**

**acc1.setBalance(1500);**

**Account acc2 = new Account();**

**acc2.setBalance(1500);**

**int amountToOperateWith = 100;**

**em.getTransaction().begin();**

**em.persist(acc1);**

**em.persist(acc2);**

**em.getTransaction().commit();**

**acc1.setBalance(acc1.getBalance() - amountToOperateWith);**

**acc2.setBalance(acc2.getBalance() + amountToOperateWith);**

**em.getTransaction().commit();**

**em.close();**

**emf.close();**

**System.out.print("Done");**

**}**

**}**

You can now test the application!

### Differences

As you can clearly see, we achieved the same results with less code by using JPA.  
Furthermore, JPA didn’t use any SQL Code, separating the concerns (two different languages) and making the code more readable for Java Programmers.