JPA with Hibernate 3
Lab Book

Table of Contents

Getting Started 3
Overview 3
Setup Checklist for JPA 3
Instructions 3
Learning More 3

Lab 1. JPA Application using Eclipse IDE 4

Lab 2. Working with JPA Association and queries 9

Appendices 11
Appendix A: Naming Conventions 11

Getting Started

Overview

This lab book is a guided tour for learning JPA with Hibernate 3. It comprises solved examples and 'To Do' assignments. Follow the steps provided in the solved examples and work out the 'To Do' assignments given.

Setup Checklist

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher (P166 recommended)
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)
- Connectivity to Oracle database

Please ensure that the following is done:

- Eclipse is installed.
- JDK 1.8 is installed.

Instructions

- For all Naming conventions, refer Appendix A. All lab assignments should adhere to naming conventions.
- Create a directory by your name in drive <drive>. In this directory, create a subdirectory jpa_assgn. For each lab exercise create a directory as lab <lab number>.

Learning More

- http://docs.oracle.com/javaee/7/api/javax/persistence
- http://www.objectdb.com/java/jpa/

Lab 1. JPA Application using Eclipse IDE

Goals	1. Setting up the Eclipse IDE environment for JPA 2. JPA CRUD application using Eclipse IDE
Time	120 minutes

1.1: Getting started with JPA in Eclipse.

Solution:

Step 1: From the File menu navigate to New, Java project.

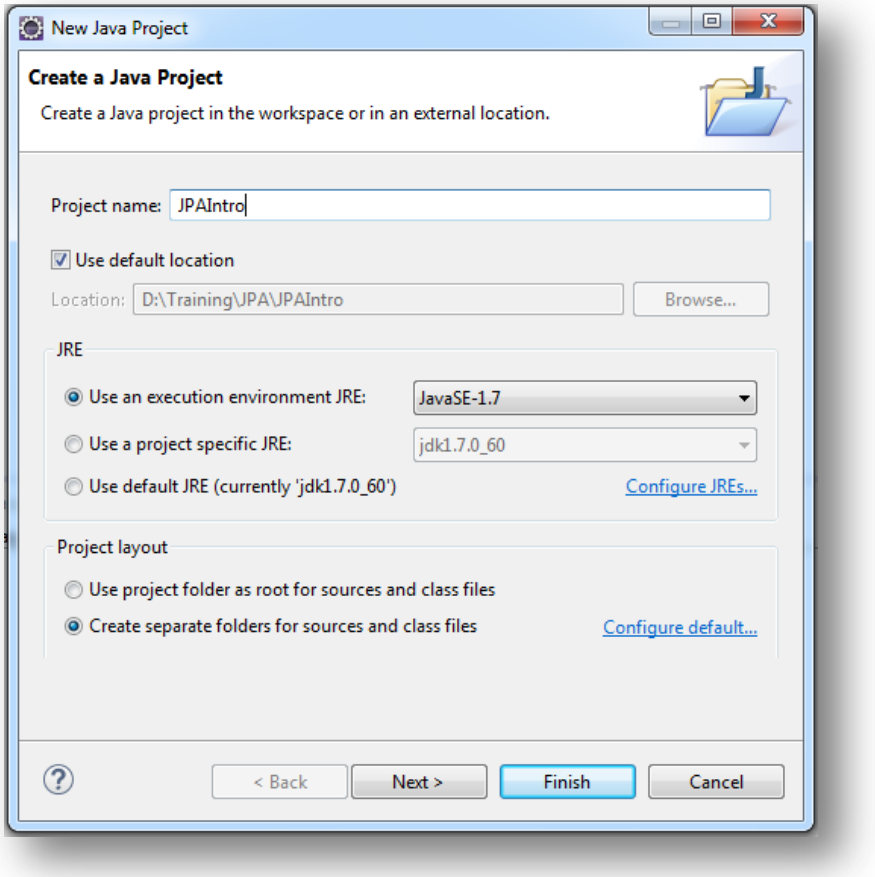


Figure 1: Create new Java project screen

Step 2: Add a new **lib** folder and paste all the dependencies provided to you.

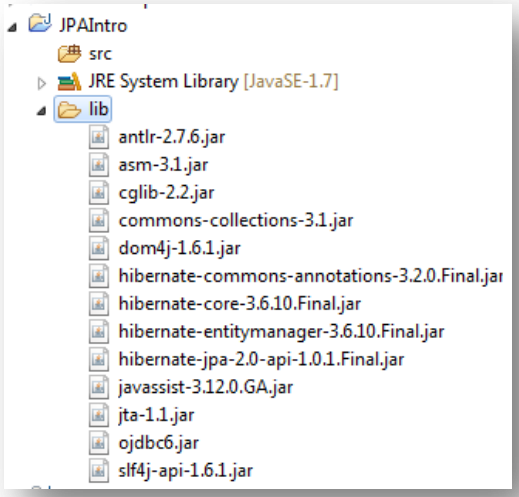


Figure 2: Required Dependencies for JPA

Step 3: Add all dependency jars to build path. Select all jars, right click and select **Build Path => Add to Build Path** from context menu. Verify a special icon on all of your dependent jars as below.

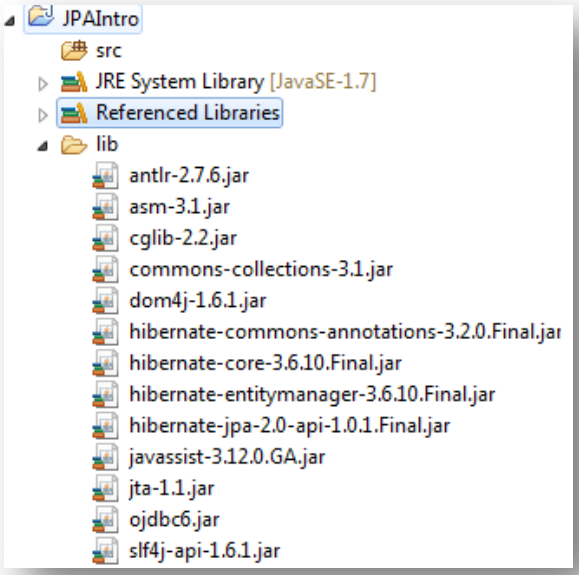


Figure 3: Dependencies added to build path

Step 4: Create a new folder called META-INF inside src directory. This folder will host persistence configuration file. i.e. persistence.xml

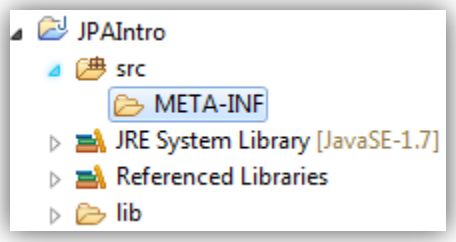


Figure 4: META-INF folder to host persistence configuration

Step 5: Right click the META-INF folder to add new xml file with name '**persistence.xml**' and paste the following contents inside file.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence

http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
      <persistence-unit name="JPA-PU" transaction-
type="RESOURCE_LOCAL">
             <provider>org.hibernate.ejb.HibernatePersistence</provider>
             <properties>
                    <property name="javax.persistence.jdbc.url"
value="<< DATABASE URL >>" />
                    <property name="javax.persistence.jdbc.user"
value="<< DATABASE USERNAME >>" />
                    <property name="javax.persistence.jdbc.password"
value="<< DATABASE PASSWORD >>" />
                    <property name="javax.persistence.jdbc.driver"
value="oracle.jdbc.driver.OracleDriver" />
             </properties>
      </persistence-unit>
</persistence>
```

Figure 5: persistence.xml

Note: Don't forget to fill property values in above file with the database details provided to you.

Step 6: Add the entity class as shown in the below code.

```
package com.cg.jpastart.entities;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Greet {

	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	private int messageId;
	private String message;

	public int getMessageId() {
		return messageId;
	}
	public void setMessageId(int messageId) {
		this.messageId = messageId;
	}
	public String getMessage() {
		return message;
	}
	public void setMessage(String message) {
		this.message = message;
	}


}
```

Step 6: Before we work on main class to test the persistence. Let us first create table in database with below SQL query.

```
CREATE TABLE greet
(messageid NUMBER(5) PRIMARY KEY,
message VARCHAR2(25));
```


Step 6: Finally add the main client class to insert record into database table.

```
package com.cg.jpastart.entities;

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

public class Client {

	public static void main(String[] args) {
		EntityManagerFactory factory = Persistence
				.createEntityManagerFactory("JPA-PU");
		EntityManager em = factory.createEntityManager();
		em.getTransaction().begin();

		Greet greet = new Greet();
		greet.setMessage("Welcome to JPA!");

		em.persist(greet);

		System.out.println("Added greeting to database.");

		em.getTransaction().commit();
		em.close();
		factory.close();
	}
}
```

<<To-do Assignments >>

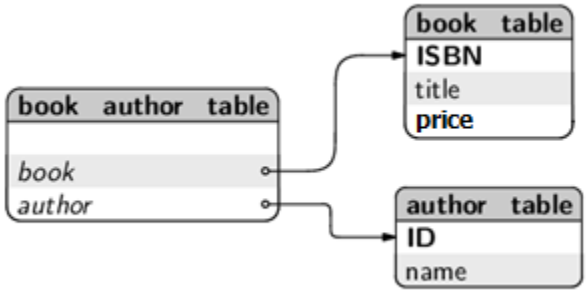
Assignment: Consider the Author class given below. Create a JPA application to perform CRUD operations (i.e. insert, update, delete) on author entity. Also display the author details based on provided author id. (**Hint**: Use find () method of EntityManager to do so) **Note:** Create required table in database before running application.

Author
authorId
firstName
middleName
lastName
phoneNo

Lab 2. Working with JPA Association and queries

Goals	• At the end of this lab session, you will be able to understand: • Implementing entity association using JPA • Working with JPA Queries
Time	120 minutes

2.1 Consider the entity relationship diagram as shown below. Identify the correct association between the book and author and implement the same using JPA. Create necessary entity classes and tables as required.



2.2: Working with JPA queries:

Extend the 2.1 case study to implement below listed queries. Write separate operations/method to implement each query.

a. Query all books in database.
b. Query all books written by given author name
c. List all books with given price range. e.g. between Rs. 500 to 1000
d. List the author name for given book id.

Appendices

Appendix A: Naming Conventions

Package names are written in all lower case to avoid conflict with the names of classes or interfaces. Companies use their reversed Internet domain name to begin their package names—for example, com.cg.mypackage for a package named mypackage created by a programmer at igate.com. Packages in the Java language itself begin with **java Or javax**

Classes and interfaces The first letter should be capitalized, and if several words are linked together to form the name, the first letter of the inner words should be uppercase (a format that's sometimes called "camelCase").

For classes, the names should typically be nouns. For example:

Dog

Account

PrintWriter

For interfaces, the names should typically be adjectives like IEmployeeService

Runnable

Serializable

Methods The first letter should be lowercase, and then normal camelCase rules should be used. In addition, the names should typically be verb-noun pairs. For example:

getBalance

doCalculation

setCustomerName

Variables Like methods, the camelCase format should be used, starting with a lowercase letter. Sun recommends short, meaningful names, which sounds good to us. Some examples:

buttonWidth

accountBalance

myString

Constants Java constants are created by marking variables static and final. They should be named using uppercase letters with underscore characters as separators:

MIN_HEIGHT

