JPA and Hibernate

By: Dalton Cave

Table of Contents

[Table of Contents i](#_Toc448669247)

[Introduction 1](#_Toc448669248)

[Prerequisites 1](#_Toc448669249)

[Who is this for? 1](#_Toc448669250)

[What is Persistence? 1](#_Toc448669251)

[Distinction between JPA and Hibernate 1](#_Toc448669252)

[Why should you care? 1](#_Toc448669253)

[How does JPA work underneath the hood? 1](#_Toc448669254)

[Notes on Hibernate 1](#_Toc448669255)

[Setting Up Hibernate 1](#_Toc448669256)

[Download Jars 1](#_Toc448669257)

[Set up Build Path in Eclipse 2](#_Toc448669258)

[Setting up config xml and HibernateUtil 2](#_Toc448669259)

[Setting up Database 2](#_Toc448669260)

[Creating Schema Manually 2](#_Toc448669261)

[Exporting Schema 2](#_Toc448669262)

[Data Access Object with xml Mapping 2](#_Toc448669263)

[Message example 2](#_Toc448669264)

[Data Access Object with Annotations 2](#_Toc448669265)

[Message example modified 2](#_Toc448669266)

[Why is inheritance a problem? 3](#_Toc448669267)

[Table Per Class 3](#_Toc448669268)

[Implicit Polymorphism 3](#_Toc448669269)

[Unions 3](#_Toc448669270)

[Table Per Subclass 3](#_Toc448669271)

[Concurrency and Hibernate 3](#_Toc448669272)

[Transactions and Locking 3](#_Toc448669273)

[Optimistic vs Pessimistic locking 3](#_Toc448669274)

[Incrementer example- note inferred annotations 3](#_Toc448669275)

[More information not covered and potential Sources 4](#_Toc448669276)

[References A](#_Toc448669277)

# Introduction

## Prerequisites

Before starting with the Java Persistence API (Henceforth JPA) there are a few concepts that you should be familiar with. Some people will prefer to jump straight in, but taking the time to build a foundation will help dampen the learning curve tremendously. There are two main prerequisites:

1. Know the basics of relational databases. Knowing about relationships and how they are represented in diagrams will help you understand how relational data is stored, and how that can lead to issues with between paradigms. Entity relationship models will be used in this document to facilitate certain concepts. Understanding them will allow you to receive the most from this document.
2. Know at least some SQL or SQL-like language. This may seem counterintuitive because the goal of JPA and Hibernate is to remove the necessity for programmers to have to directly use SQL. However, knowing SQL will allow you to see why someone would want to use JPA and Hibernate and also allow you to better understand some of the more complex task that you can achieve with them.

## Who is this for?

JPA and Hibernate are useful for Java programmers who deal with many, often complex, database interactions, and would like to do so in an object oriented manner. Later, we will discuss why programmers should want to use JPA and Hibernate, as well as situations where they would not necessarily be better than SQL.

## What is Persistence?

In order to understand JPA, we must first know what is meant by *persistence* in computer science. Wikipedia defines persistence: “Persistence refers to a characteristic of state that outlives the process that created it”. Often times this means using a database, and when the term persistence is used in this document we will be specifically referring to storing data in a database.

Another important phrase used is Object Relational Mapping (Henceforth ORM). Since Java code is Object Oriented, and most databases are not, we use the idea of mapping to transpose data back and forth between these two paradigms. This is the driving feature of using JPA and Hibernate.

## 

## Distinction between JPA and Hibernate

We keep talking about JPA and Hibernate, but what is the difference? JPA is Java’s API for dealing with databases using ORM. It works by translating Java objects into SQL queries, allowing the programmer to further abstract themselves from the persistence of data. Hibernate, on the other hand, is an implementation of JPA. Hibernate does a lot of the heavy lifting and keeps the developers focused on the more important logic that happens with the objects.

Why shouldn’t we make our own implementation of JPA? Besides the time and effort that will go into giving you something that already exists with Hibernate, Hibernate has been in development for years, and is used and tested by many. By using Hibernate you are decreasing the number of bugs that could be introduced by making your own data access implementation using JPA. In addition to this, using Hibernate does not restrict you from using JPA, so if you do happen to find something that you cannot do with Hibernate, you still have JPA as an option. Because Hibernate already provides an implementation of JPA, the example code and discussion will be based around Hibernate.

## 

## Why should you care?

There are many benefits to using JPA and Hibernate. First off, it greatly diminishes the time and effort that it takes to implement basic Create, Read, Update, and Delete. Another benefit is that after the setup of the Data Access Object, you only need to know about the Java object. You don’t need to know anything about the Database. The Object Oriented Paradigm is distinctly different than that of the Relational Paradigm. Inheritance is particularly different to model in a Java program than in a relational database. With some help, Hibernate can automatically map these schemas and make it significantly easier to persist complex hierarchies. Hibernate is often faster than hand coded SQL because it can perform optimizations. Show an example of an SQL query with JDBC and Hibernate. Pull some quotes from Java Persistence Book. Times not to use Hibernate, non-complex, quick applications, rare database access. Doesn’t mean you can’t use them, but you save time exponentially.

## 

## Notes on Hibernate

Hibernate has been in development since 2001 and is still actively being improved today. It implements JPA 2.1. It supports 25 different databases out of the box as long as you have the JDBC drivers. With a little extra configuration, you can use pretty much any relational database. For my examples in this code I am using a MySQL database running locally on my computer.

# Setting Up Hibernate

## Download Jars

The first step to making a project with Hibernate is to download the necessary files. This will include the Hibernate files, as well as the JDBC driver for your database. The hibernate files can be found at <http://hibernate.org/orm/downloads/>. For my demonstration I will be using a MySQL server. If you would like to follow along with MySQL you can acquire the JDBC driver at <https://dev.mysql.com/downloads/connector/j/>.

## 

## Set up Build Path in Eclipse

Before we can use the jars that we downloaded we must first add them to our build path. If you skip this step then the compiler will complain and you will not be able to actually run the hibernate code. There are several ways that you can go about this. If you are familiar with ANT then you could add the necessary files to your ANT xml. If you are using the eclipse IDE a quick way to add the jars to the build path of your project is to right click on

-Add external jars is a quick way to add to the project

## Setting up config xml and HibernateUtil

-Walk through the concepts of the config xml, and reference the code

-Talk about creating sessions and SessionFactory and why having a class like HibernateUtil

## Setting up Database

### Creating Schema Manually

### Exporting Schema

# Data Access Object with xml Mapping

## Message example

-Message mapping xml example walkthrough

-Changes to config xml

-Create and Read

-Reference Code

# Data Access Object with Annotations

## Message example modified

-Annotations and their meaning

-Ability to mix and match annotations default behavior

-Criteria for READ

-Delete

-Changes to config xml

-Reference CodeInheritance Strategies

# Why is inheritance a problem?

## Table Per Class

### Implicit Polymorphism

### Unions

## Table Per Subclass

# Concurrency and Hibernate

-How does Hibernate deal with concurrency?

-Marking immutable

## Transactions and Locking

-Obtains lock per transactions, different levels of locking know as isolation levels

## Optimistic vs Pessimistic locking

-Optimistic assumes conflicts are rare and allows rollback via versioning

-Pessimistic locks database access and assumes that conflicts are common. Often not a

Good strategy, avoid if possible, essentially forces serialization

## Incrementer example- note inferred annotations

-Update

# More information not covered and potential Sources

-Custom Types

-Caching

-HQL & JPAQL

-So called non-transactional data access

-Conversations

-Filtering Collections

# References