Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Know JavaScript?
Build Mobile Apps
Using Your Web Skills
Start
Teler

ABOUT THEODORA FRAGKOULI

Theodora has graduated from Computer Engineering and Informatics Department in the University of Patras. She also holds a Master degree in Economics from the National and Technical University of Athens. During her studies she has been involved with a large number of projects ranging from programming and software engineering to telecommunications, hardware design and analysis. She works as a junior Software Engineer in the telecommunications sector where she is mainly involved with projects based on Java and Big Data technologies.

Hibernate JPA DAO Example

Posted by: Theodora Fragkouli in hibernate November 5th, 2014

This is an example of how to create Data Access Objects (DAOs), making use of the Hibernate implementation for the Java Persistence API (JPA) specification. Hibernate is an object-relational mapping library for Java, that provides a framework for mapping an object-oriented domain model to a traditional relational database.

When an application interacts with a database, it is common pattern to separate all the low level data access operations from high level business services. This can be achieved using DAOs, which are objects that provide abstract interfaces to the database. DAOs may be used from services in higher layers of the application, thus connecting a service layer and the database.

Below, we will create a DAO class (Data Access Layer) and will call its interface in another class (Service Layer) to create, read, update and delete rows (CRUD operations) from a database table.

Tip

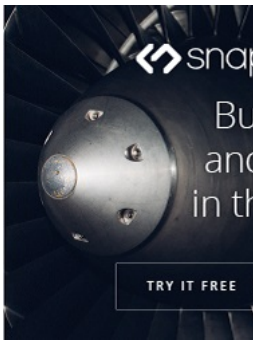
You may skip project creation and jump directly to the **beginning of the example** below.

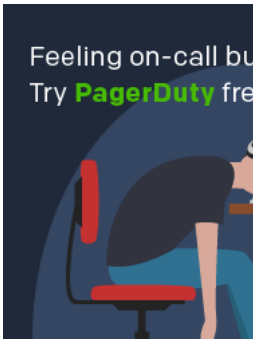
Our preferred development environment is Eclipse. We are using Eclipse Juno (4.2) version, along with Maven Integration plugin version 3.1.0. You can download Eclipse from here and Maven Plugin for Eclipse from here. The installation of Maven plugin for Eclipse is out of the scope of this tutorial and will not be discussed. We are also using the JDK 7_u_21. The Hibernate version is 4.3.6, and the database used in the example is MySQL Database Server 5.6.

Let's begin,

1. Create a new Maven project

Go to File -> Project ->Maven -> Maven Project.





NEWSLETTER

**104676 insiders are al
weekly updates and compl
whitepapers!
Join them now to ga
exclusive access to
news in the Java world, a
insights about Android, Sc
and other related technolo**

Email address:

Your email address

Sign up

JOIN US

**With 1
monthl
and ov
we are
the top
sites a
being o
partne
you to j
have a blog with unique and
content then you should chec
partners program. You can a
writer for Java Code Geeks
writing skills!**

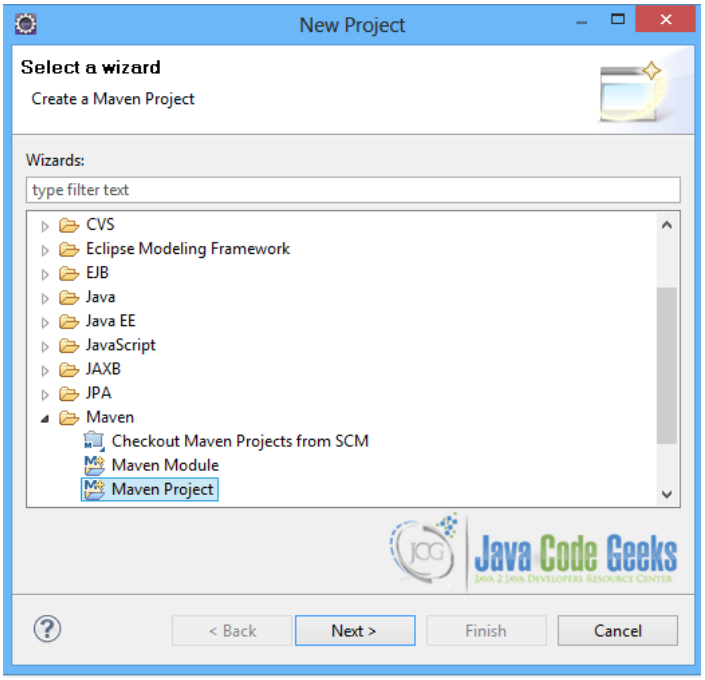


Figure 1: New Maven Project – step 1

In the "Select project name and location" page of the wizard, make sure that "Create a simple project (skip archetype selection)" option is **checked**, hit "Next" to continue with default values.

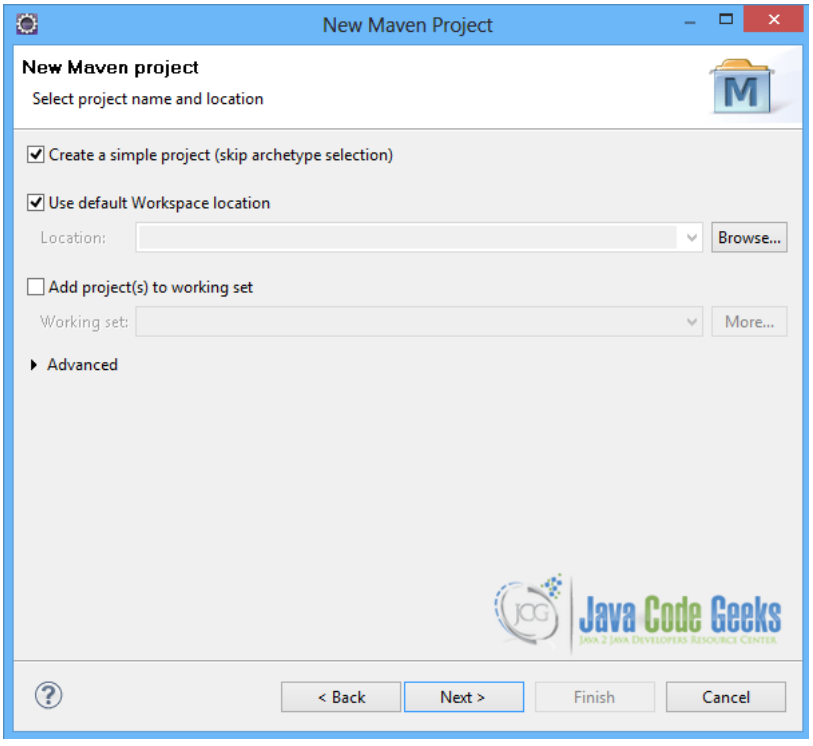


Figure 2: New Maven Project 2

In the "Enter an artifact id" page of the wizard, you can define the name and main package of your project. We will set the "Group Id" variable to

```
"com.javacodegeeks.snippets.enterprise"
```

and the "Artifact Id" variable to

```
"hibernateexample"
```

. The aforementioned selections compose the main project package as

```
"com.javacodegeeks.snippets.enterprise.hibernateexample"
```

and the project name as

```
"hibernateexample"
```

. Hit "Finish" to exit the wizard and to create your project.

CAREER OPPORTUNITIES

Application Developer
JPMorgan Chase - United States
Enterprise Software Engineer
Boeing - Bellevue, WA
Java Developer
Bank of America - New York, NY
Sr.Java Developer
Infojini Inc - Washington, DC
Java Developer
Cognizant - Cupertino, CA
Back End Engineer
Transmosis - Santa Monica, CA
Java Developer
Highmark Companies - United States
Java Developer
Reliance One - Farmington, MI
Java Developer
Cognizant - San Francisco, CA
Software Development Engine
Expedia, Inc. - Bellevue, WA
1 2 3 4 5 6 7 8 9 10 Next »
What:
title, keywords
Find Jobs

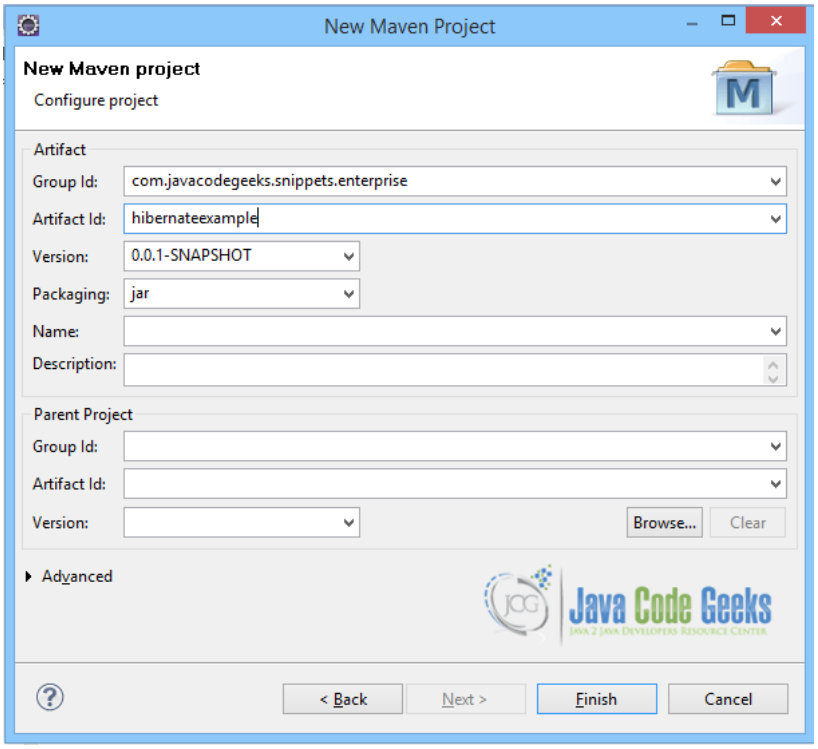


Figure 3: hibernateexample

The Maven project structure is shown below:

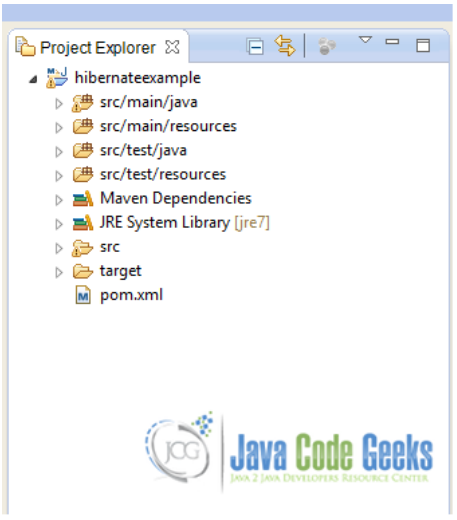


Figure 4: Project Structure

It consists of the following folders:

- /src/main/java folder, that contains source files for the dynamic content of the application,
- /src/test/java folder contains all source files for unit tests,
- /src/main/resources folder contains configurations files,
- /target folder contains the compiled and packaged deliverables,
- the pom.xml is the project object model (POM) file. The single file that contains all project related configuration.

2. Add hibernate 4.3.6 dependency

You can add all the necessary dependencies in Maven's

```
pom.xml
```

file, by editing it at the "Pom.xml" page of the POM editor. Apart from

```
hibernate
```

dependency, we will also need the

```
mysql-connector-java
```

package.

pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.javacodegeeks.snippets.enterprise</groupId>
    <artifactId>hibernateexample</artifactId>
    <version>0.0.1-SNAPSHOT</version>


    <dependencies>

        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>4.3.6.Final</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.6</version>
        </dependency>
    </dependencies>
</project>
```

As you can see Maven manages library dependencies declaratively. A local repository is created (by default under

```
{user_home}/.m2
```

folder) and all required libraries are downloaded and placed there from public repositories. Furthermore intra – library dependencies are automatically resolved and manipulated.

3. Create the entity class

```
Book.java
```

class is the entity class which uses some basic Hibernate JPA annotations to be mapped to

```
BOOK
```

table in the database.

Book.java

```
package com.javacodegeeks.snippets.enterprise.hibernate.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "book")
public class Book {

    @Id
    @Column(name = "id")
    private String id;

    @Column(name = "title")
    private String title;

    @Column(name= "author")
    String author;

    public Book() {
    }

    public Book(String id, String title, String author) {
        this.id = id;
        this.title = title;
        this.author = author;
    }

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    @Override
```

```
61      public String toString() {
62          return "Book: " + this.id + ", " + this.title + ", " + this.author;
63      }
64
65  }
```

4. Create the DAO class

`BookDao.java`

class is the Dao class, which contains all the basic CRUD methods to interact with the database.

First of all,

`getSessionFactory()`

is a

`static`

method that provides a

`SessionFactory`

, the creator of

`Sessions`

, the basic interfaces between a Java application and Hibernate. The

`SessionFactory`

is built with the

`StandardServiceRegistryBuilder`

, making use of

`Configuration`

. The

`Configuration`

is where we can specify properties and mapping documents to be used when creating a

`SessionFactory`

.

So, every method that interacts with the database gets a

`Session`

, making use of the

`getSessionFactory()`

.

Two basic methods are used to get a

`Session`

from the

`SessionFactory`

, the

`openCurrentSession()`

and

`openCurrentSessionwithTransaction()`

. Both methods use the

`openSession()`

API method of

`SessionFactory`

. But the second one also opens a new transaction, making use of the

`beginTransaction()`

API method of

`Session`

.

Two basic methods are also used to close the

```
Session
```

, the

```
closeCurrentSession
```

and

```
closeCurrentSessionwithTransaction()
```

. Both methods use the

```
session.close()
```

API method of

```
Session
```

to close the

```
Session
```

, but the second method first commits the transaction, using

```
getTransaction().commit()
```

API method.

The basic CRUD methods to interact with a database are **Create**, **Read**, **Update** and **Delete**.

Create is done in

```
persist(Book entity)
```

method, with

```
save(Object object)
```

API method of

```
Session
```

, that persists an entity to the database.

Read is performed both in

```
findById(String id)
```

and in

```
findAll()
```

methods.

```
findById
```

method uses

```
get(Class theClass, Serializable id)
```

API method of

```
Session
```

to retrieve an object by its id, whereas

```
findAll
```

creates a new Query with a String SQL query, to get all rows of the table in a list.

Update is easily done in

```
update(Book entity)
```

method that uses

```
update(Object object)
```

API method of

```
Session
```

.

Delete is performed in

```
delete(Book entity)
```

and

```
deleteAll()
```

methods, using the

```
findById(String id)
```

and

```
findAll()
```

methods respectively to retrieve the objects from the database and then using

```
delete(Object object)
```

API method of

```
Session
```

.

BookDao.java

```
01 package com.javacodegeeks.snippets.enterprise.hibernate.dao;
02
03 import java.util.List;
04
05 import org.hibernate.Session;
06 import org.hibernate.SessionFactory;
07 import org.hibernate.Transaction;
08 import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
09 import org.hibernate.cfg.Configuration;
10
11 import com.javacodegeeks.snippets.enterprise.hibernate.model.Book;
12
13 public class BookDao implements BookDaoInterface<Book, String> {
14
15     private Session currentSession;
16
17     private Transaction currentTransaction;
18
19     public BookDao() {
20     }
21
22     public Session openCurrentSession() {
23         currentSession = getSessionFactory().openSession();
24         return currentSession;
25     }
26
27     public Session openCurrentSessionwithTransaction() {
28         currentSession = getSessionFactory().openSession();
29         currentTransaction = currentSession.beginTransaction();
30         return currentSession;
31     }
32
33     public void closeCurrentSession() {
34         currentSession.close();
35     }
36
37     public void closeCurrentSessionwithTransaction() {
38         currentTransaction.commit();
39         currentSession.close();
40     }
41
42     private static SessionFactory getSessionFactory() {
43         Configuration configuration = new Configuration().configure();
44         StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder()
45                 .applySettings(configuration.getProperties());
46         SessionFactory sessionFactory = configuration.buildSessionFactory(builder.build());
47         return sessionFactory;
48     }
49
50     public Session getCurrentSession() {
51         return currentSession;
52     }
53
54     public void setCurrentSession(Session currentSession) {
55         this.currentSession = currentSession;
56     }
57
58     public Transaction getCurrentTransaction() {
59         return currentTransaction;
60     }
61
62     public void setCurrentTransaction(Transaction currentTransaction) {
63         this.currentTransaction = currentTransaction;
64     }
65
66     public void persist(Book entity) {
67         getCurrentSession().save(entity);
68     }
69
70     public void update(Book entity) {
71         getCurrentSession().update(entity);
72     }
73
74     public Book findById(String id) {
75         Book book = (Book) getCurrentSession().get(Book.class, id);
76         return book;
77     }
78
79     public void delete(Book entity) {
80         getCurrentSession().delete(entity);
81     }
82
83     @SuppressWarnings("unchecked")
84     public List<Book> findAll() {
85         List<Book> books = (List<Book>) getCurrentSession().createQuery("from Book").list();
86         return books;
87     }
```

```
88
89      public void deleteAll() {
90          List<Book> entityList = findAll();
91          for (Book entity : entityList) {
92              delete(entity);
93          }
94      }
95  }
```

Below is the DAO interface that contains all methods that we want to be exposed in the Service layer.

BookDaoInterface.java

```
01  package com.javacodegeeks.snippets.enterprise.hibernate.dao;
02
03  import java.io.Serializable;
04  import java.util.List;
05
06  public interface BookDaoInterface<T, Id extends Serializable> {
07
08      public void persist(T entity);
09
10      public void update(T entity);
11
12      public T findById(Id id);
13
14      public void delete(T entity);
15
16      public List<T> findAll();
17
18      public void deleteAll();
19
20  }
```

5. Create the Service class

`BookService.java`

class is the service which makes use of the DAO object to interact with the database. The DAO object is a

`static`

field in the service, initialized in the service constructor. So, when a new service instance is created, a new DAO instance will also be created.

In each one of the service methods, the

`bookDao`

object is used to open/close a session or a session with transaction, and to perform each one of the CRUD actions described above. In this layer all the transactions are handled. For example,

`persist`

,

`update`

and

`delete`

methods must follow the

`openSessionWithTransaction()`

method, whereas,

`findById`

and

`findAll`

methods only need the

`openSession()`

method.

BookService.java

```
01  package com.javacodegeeks.snippets.enterprise.hibernate.service;
02
03  import java.util.List;
04
05  import com.javacodegeeks.snippets.enterprise.hibernate.dao.BookDao;
06  import com.javacodegeeks.snippets.enterprise.hibernate.model.Book;
07
08  public class BookService {
09
10      private static BookDao bookDao;
11
12      public BookService() {
13          bookDao = new BookDao();
14      }
15
16      public void persist(Book entity) {
17          bookDao.openCurrentSessionwithTransaction();
```

```
        bookDao.persist(entity);
        bookDao.closeCurrentSessionwithTransaction();
    }

    public void update(Book entity) {
        bookDao.openCurrentSessionwithTransaction();
        bookDao.update(entity);
        bookDao.closeCurrentSessionwithTransaction();
    }

    public Book findById(String id) {
        bookDao.openCurrentSession();
        Book book = bookDao.findById(id);
        bookDao.closeCurrentSession();
        return book;
    }

    public void delete(String id) {
        bookDao.openCurrentSessionwithTransaction();
        Book book = bookDao.findById(id);
        bookDao.delete(book);
        bookDao.closeCurrentSessionwithTransaction();
    }

    public List<Book> findAll() {
        bookDao.openCurrentSession();
        List<Book> books = bookDao.findAll();
        bookDao.closeCurrentSession();
        return books;
    }

    public void deleteAll() {
        bookDao.openCurrentSessionwithTransaction();
        bookDao.deleteAll();
        bookDao.closeCurrentSessionwithTransaction();
    }

    public BookDao bookDao() {
        return bookDao;
    }
}
```

6. Configure hibernate

The

`hibernate.cfg.xml`

file shown below is where all configuration needed for the interaction with the database is set. The database that is used is defined here, as well as the database user credentials. The dialect is set to

`MySQL`

, and the driver is the

`com.mysql.jdbc.Driver`

. There is also a

`mapping`

attribute, where the entity class is defined.

You can also set specific database options here, such as whether the schema will be created or just updated, every time the

`sessionFactory`

is created. This is configured in the

`hibernate.hbm2ddl.auto`

property, which is set to

`update`

. So the schema is only updated. If this property is set to

`create`

, then every time we run our application, the schema will be re-created, thus deleting previous data. Another property set here is the

`show_sql`

, which specifies whether the sql queries will be shown in the console or the logger. Finally, the

`hibernate.current_session_context_class`

is set to

`thread`

, meaning that the

`SessionFactory`

will bind the

`Session`

to the thread from which

```
openSession()
```

method is called.

hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>
   <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
   <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
   <property name="hibernate.connection.url">jdbc:mysql://localhost/library</property>
   <property name="hibernate.connection.username">root</property>
   <property name="hibernate.connection.password">root</property>
   <property name="hibernate.hbm2ddl.auto">update</property>
   <property name="show_sql">false</property>
   <property name="hibernate.current_session_context_class">thread</property>
   <mapping class="com.javacodegeeks.snippets.enterprise.hibernate.model.Book"/>
</session-factory>
</hibernate-configuration>
```

7. Run the Application

In order to run the example, first create a

```
library
```

database and add a

```
book
```

table, using the SQL statement below:

Create Book table statement

```
CREATE TABLE `library`.`book` (
id VARCHAR(50) NOT NULL,
title VARCHAR(20) default NULL,
author VARCHAR(50) default NULL,
PRIMARY KEY (id)
);
```

Then, run the following application. It creates a new

```
BookService
```

instance, which also creates its own

```
bookDao
```

instance to interact with the database.

App.java

```
package com.javacodegeeks.snippets.enterprise.hibernate;

import java.util.List;

import com.javacodegeeks.snippets.enterprise.hibernate.model.Book;
import com.javacodegeeks.snippets.enterprise.hibernate.service.BookService;

public class App {

    public static void main(String[] args) {
        BookService bookService = new BookService();
        Book book1 = new Book("1", "The Brothers Karamazov", "Fyodor Dostoevsky");
        Book book2 = new Book("2", "War and Peace", "Leo Tolstoy");
        Book book3 = new Book("3", "Pride and Prejudice", "Jane Austen");
        System.out.println("*** Persist - start ***");
        bookService.persist(book1);
        bookService.persist(book2);
        bookService.persist(book3);
        List<Book> books1 = bookService.findAll();
        System.out.println("Books Persisted are :");
        for (Book b : books1) {
            System.out.println("-" + b.toString());
        }
        System.out.println("*** Persist - end ***");
        System.out.println("*** Update - start ***");
        book1.setTitle("The Idiot");
        bookService.update(book1);
        System.out.println("Book Updated is =>" +bookService.findById(book1.getId()).toString());
        System.out.println("*** Update - end ***");
        System.out.println("*** Find - start ***");
        String id1 = book1.getId();
        Book another = bookService.findById(id1);
        System.out.println("Book found with id " + id1 + " is =>" + another.toString());
        System.out.println("*** Find - end ***");
        System.out.println("*** Delete - start ***");
        String id3 = book3.getId();
        bookService.delete(id3);
        System.out.println("Deleted book with id " + id3 + ".");
        System.out.println("Now all books are " + bookService.findAll().size() + ".");
        System.out.println("*** Delete - end ***");
        System.out.println("*** FindAll - start ***");
        List<Book> books2 = bookService.findAll();
```

```
        System.out.println("Books found are :");
        for (Book b : books2) {
            System.out.println("-" + b.toString());
        }
        System.out.println("*** FindAll - end ***");
        System.out.println("*** DeleteAll - start ***");
        bookService.deleteAll();
        System.out.println("Books found are now " + bookService.findAll().size());
        System.out.println("*** DeleteAll - end ***");
         System.exit(0);
    }
}
```

When you run the application, you will see that all basic CRUD actions are performed. Three books are created, then one is updated, then one is deleted, and finally all books are deleted.

Output

```
*** Persist - start ***
Books Persisted are :
-Book: 1, The Brothers Karamazov, Fyodor Dostoevsky
-Book: 2, War and Peace, Leo Tolstoy
-Book: 3, Pride and Prejudice, Jane Austen
*** Persist - end ***
*** Update - start ***
Book Updated is =>Book: 1, The Idiot, Fyodor Dostoevsky
*** Update - end ***
*** Find - start ***
Book found with id 1 is =>Book: 1, The Idiot, Fyodor Dostoevsky
*** Find - end ***
*** Delete - start ***
Deleted book with id 3.
Now all books are 2.
*** Delete - end ***
*** FindAll - start ***
Books found are :
-Book: 1, The Idiot, Fyodor Dostoevsky
-Book: 2, War and Peace, Leo Tolstoy
*** FindAll - end ***
*** DeleteAll - start ***
Books found are now 0
*** DeleteAll - end ***
```

Tip

You can take a look on another implementation of **Hibernate JPA DAOs**, using **Spring** integration **here**.

8. Download the Eclipse Project

This was an example of how to create JPA DAOs using Hibernate.

Download

You can download the full source code of this example here: **HibernateJPADAOExample.zip**

Tagged with: HIBERNATE JPA MYSQL

Do you want to know how to develop your skillset to become a Java Rockstar?

Subscribe to our newsletter to start Rocking right now!
To get you started we give you our best selling eBooks for FREE!

1. JPA Mini Book
2. JVM Troubleshooting Guide
3. JUnit Tutorial for Unit Testing
4. Java Annotations Tutorial
5. Java Interview Questions
6. Spring Interview Questions
7. Android UI Design

and many more ....

Email address:

Your email address

Sign up

KNOWLEDGE BASE

- Academy
- Library
- News
- Resources
- Tutorials
- Whitepapers

THE CODE GEEKS NETWORK

- .NET Code Geeks
- Java Code Geeks
- Web Code Geeks

HALL OF FAME

- Android Alert Dialog Example
- Android OnClickListener Example
- How to convert Character to String and a String to Character Array in Java
- Java Inheritance example
- Java write to File Example
- java.io.FileNotFoundException – How to solve File Not Found Exception
- java.lang.arrayindexoutofboundsexce – How to handle Array Index Out Of Bounds Exception
- java.lang.NoClassDefFoundError – How to solve No Class Def Found Error
- JSON Example With Jersey + Jackson
- Spring JdbcTemplate Example

ABOUT JAVA CODE GEEKS

JCGs (Java Code Geeks) is an independent online community f creating the ultimate Java to Java developers resource center the technical architect, technical team lead (senior developer) manager and junior developers alike. JCGs serve the Java, SO Telecom communities with daily news written by domain expe tutorials, reviews, announcements, code snippets and open so

DISCLAIMER

All trademarks and registered trademarks appearing on Exam Geeks are the property of their respective owners. Java is a t registered trademark of Oracle Corporation in the United State countries. Examples Java Code Geeks is not connected to Orac and is not sponsored by Oracle Corporation.

Examples Java Code Geeks and all content copyright © 2010-2015, Exelixis Media P.C. | Terms of Use | Privacy Policy | Contact