**Exercise 001:**

Let's consider a simple example of a one-to-many relationship between two entities: **Department** and **Employee**. Each department can have multiple employees.

1. Define the **Department** and **Employee** entities with a one-to-many relationship.
2. Create a sample Java program to insert a department and its employees into the database.
3. Retrieve and display the employees of a specific department.

Answer:

Entity classes:

@Entity

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "department", cascade = CascadeType.ALL)

private List<Employee> employees;

// Constructors, getters, and setters

}

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

// Constructors, getters, and setters

}

**Inserting a department and employees:**

public class Main {

public static void main(String[] args) {

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Department.class)

.addAnnotatedClass(Employee.class)

.buildSessionFactory();

Session session = factory.getCurrentSession();

try {

session.beginTransaction();

Department department = new Department("HR");

Employee employee1 = new Employee("John");

Employee employee2 = new Employee("Jane");

department.addEmployee(employee1);

department.addEmployee(employee2);

session.save(department);

session.getTransaction().commit();

} finally {

factory.close();

}

}

}

**Retrieving employees of a department:**

public class Main {

public static void main(String[] args) {

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Department.class)

.addAnnotatedClass(Employee.class)

.buildSessionFactory();

Session session = factory.getCurrentSession();

try {

session.beginTransaction();

Department department = session.get(Department.class, 1); // Replace 1 with the department's ID you want to retrieve

List<Employee> employees = department.getEmployees();

for (Employee employee : employees) {

System.out.println("Employee: " + employee.getName());

}

session.getTransaction().commit();

} finally {

factory.close();

}

}

}

**Exercise002:**

Let's consider a simple example of a one-to-many relationship between two entities: **Course** and **Student**. Each course can have multiple students.

1. Define the **Course** and **Student** entities with a one-to-many relationship.
2. Create a sample Java program to insert courses and students into the database.
3. Retrieve and display the students enrolled in a specific course.

Answer:

**Entity classes:**

@Entity

public class Course {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "course", cascade = CascadeType.ALL)

private List<Student> students;

// Constructors, getters, and setters

}

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@ManyToOne

@JoinColumn(name = "course\_id")

private Course course;

// Constructors, getters, and setters

}

**Inserting courses and students:**

public class Main {

public static void main(String[] args) {

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Course.class)

.addAnnotatedClass(Student.class)

.buildSessionFactory();

Session session = factory.getCurrentSession();

try {

session.beginTransaction();

Course course = new Course("History");

Student student1 = new Student("Alice");

Student student2 = new Student("Bob");

course.addStudent(student1);

course.addStudent(student2);

session.save(course);

session.getTransaction().commit();

} finally {

factory.close();

}

}

}

**Retrieving students of a course:**

public class Main {

public static void main(String[] args) {

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Course.class)

.addAnnotatedClass(Student.class)

.buildSessionFactory();

Session session = factory.getCurrentSession();

try {

session.beginTransaction();

Course course = session.get(Course.class, 1); // Replace 1 with the course's ID you want to retrieve

List<Student> students = course.getStudents();

for (Student student : students) {

System.out.println("Student: " + student.getName());

}

session.getTransaction().commit();

} finally {

factory.close();

}

}

}

**hibernate.cfg.xml**

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- Database connection settings -->

<property name="hibernate.connection.driver\_class">your.database.driver.class</property>

<property name="hibernate.connection.url">jdbc:your\_database\_url</property>

<property name="hibernate.connection.username">your\_username</property>

<property name="hibernate.connection.password">your\_password</property>

<!-- JDBC connection pool settings -->

<property name="hibernate.c3p0.min\_size">5</property>

<property name="hibernate.c3p0.max\_size">20</property>

<property name="hibernate.c3p0.timeout">300</property>

<property name="hibernate.c3p0.max\_statements">50</property>

<property name="hibernate.c3p0.idle\_test\_period">3000</property>

<!-- Specify dialect -->

<property name="hibernate.dialect">org.hibernate.dialect.YourDialect</property>

<!-- Echo all executed SQL to stdout -->

<property name="hibernate.show\_sql">true</property>

<!-- Drop and re-create the database schema on startup -->

<property name="hibernate.hbm2ddl.auto">update</property>

<!-- Entity classes -->

<mapping class="com.example.Course" />

<mapping class="com.example.Student" />

</session-factory>

</hibernate-configuration>

**Exercise003:**

Let's create a more comprehensive example that includes all CRUD operations (Create, Read, Update, Delete) for a simple one-to-many relationship between **Author** and **Book** entities.

Authors can write multiple books, and each book is associated with one author.

**Answer:**

**Entity Classes:**

1. Author.java:

@Entity

public class Author {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "author", cascade = CascadeType.ALL)

private List<Book> books = new ArrayList<>();

// Constructors, getters, and setters

}

1. Book.java:

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

@ManyToOne

@JoinColumn(name = "author\_id")

private Author author;

// Constructors, getters, and setters

}

**Hibernate Configuration (hibernate.cfg.xml):**

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- Database connection settings -->

<property name="hibernate.connection.driver\_class">your.database.driver.class</property>

<property name="hibernate.connection.url">jdbc:your\_database\_url</property>

<property name="hibernate.connection.username">your\_username</property>

<property name="hibernate.connection.password">your\_password</property>

<!-- JDBC connection pool settings -->

<property name="hibernate.c3p0.min\_size">5</property>

<property name="hibernate.c3p0.max\_size">20</property>

<property name="hibernate.c3p0.timeout">300</property>

<property name="hibernate.c3p0.max\_statements">50</property>

<property name="hibernate.c3p0.idle\_test\_period">3000</property>

<!-- Specify dialect -->

<property name="hibernate.dialect">org.hibernate.dialect.YourDialect</property>

<!-- Echo all executed SQL to stdout -->

<property name="hibernate.show\_sql">true</property>

<!-- Drop and re-create the database schema on startup -->

<property name="hibernate.hbm2ddl.auto">update</property>

<!-- Entity classes -->

<mapping class="com.example.Author" />

<mapping class="com.example.Book" />

</session-factory>

</hibernate-configuration>

Replace the placeholders (**your.database.driver.class**, **jdbc:your\_database\_url**, **your\_username**, **your\_password**, **org.hibernate.dialect.YourDialect**) with your actual database and Hibernate configuration details.

**CRUD Operations:**

Here's an example of CRUD operations:

1. **Create (Insert) Author and Books:**

public class Main {

public static void main(String[] args) {

// Create and configure the SessionFactory as shown in previous examples

try (Session session = factory.getCurrentSession()) {

session.beginTransaction();

Author author = new Author("J.K. Rowling");

Book book1 = new Book("Harry Potter and the Sorcerer's Stone", author);

Book book2 = new Book("Harry Potter and the Chamber of Secrets", author);

author.addBook(book1);

author.addBook(book2);

session.save(author);

session.getTransaction().commit();

}

}

}

1. **Read (Retrieve) Author and Books:**

public class Main {

public static void main(String[] args) {

// Create and configure the SessionFactory as shown in previous examples

try (Session session = factory.getCurrentSession()) {

session.beginTransaction();

Author author = session.get(Author.class, 1); // Replace 1 with the author's ID to retrieve

List<Book> books = author.getBooks();

for (Book book : books) {

System.out.println("Book Title: " + book.getTitle());

}

session.getTransaction().commit();

}

}

}

1. **Update Author and Books:**

public class Main {

public static void main(String[] args) {

// Create and configure the SessionFactory as shown in previous examples

try (Session session = factory.getCurrentSession()) {

session.beginTransaction();

Author author = session.get(Author.class, 1); // Replace 1 with the author's ID to update

author.setName("New Author Name");

session.getTransaction().commit();

}

}

}

1. **Delete Author and Books:**

public class Main {

public static void main(String[] args) {

// Create and configure the SessionFactory as shown in previous examples

try (Session session = factory.getCurrentSession()) {

session.beginTransaction();

Author author = session.get(Author.class, 1); // Replace 1 with the author's ID to delete

session.delete(author);

session.getTransaction().commit();

}

}

}

These CRUD operations demonstrate the basics of working with a one-to-many relationship in Hibernate. Ensure that you adapt the code to your specific needs and update the Hibernate configuration accordingly.

**Exercise004:**

In this exercise, we'll create a simple library management system with two entities: Library and Book, where each library can have multiple books.

Define the Library and Book entities with a one-to-many relationship.

Implement CRUD operations for both libraries and books using Hibernate. The operations should include creating, reading, updating, and deleting records.

Create a Java program to demonstrate these CRUD operations.

**Answers:**

Let's start with the entity classes:

**Library.java:**

@Entity

public class Library {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "library", cascade = CascadeType.ALL)

private List<Book> books = new ArrayList<>();

// Constructors, getters, and setters

}

**Book.java:**

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

@ManyToOne

@JoinColumn(name = "library\_id")

private Library library;

// Constructors, getters, and setters

}

**Next, create a hibernate.cfg.xml file as previously shown with the appropriate database configuration and entity mappings.**

**hibernate.cfg.xml**

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- Database connection settings -->

<property name="hibernate.connection.driver\_class">your.database.driver.class</property>

<property name="hibernate.connection.url">jdbc:your\_database\_url</property>

<property name="hibernate.connection.username">your\_username</property>

<property name="hibernate.connection.password">your\_password</property>

<!-- JDBC connection pool settings -->

<property name="hibernate.c3p0.min\_size">5</property>

<property name="hibernate.c3p0.max\_size">20</property>

<property name="hibernate.c3p0.timeout">300</property>

<property name="hibernate.c3p0.max\_statements">50</property>

<property name="hibernate.c3p0.idle\_test\_period">3000</property>

<!-- Specify dialect -->

<property name="hibernate.dialect">org.hibernate.dialect.YourDialect</property>

<!-- Echo all executed SQL to stdout -->

<property name="hibernate.show\_sql">true</property>

<!-- Drop and re-create the database schema on startup -->

<property name="hibernate.hbm2ddl.auto">update</property>

<!-- Entity classes -->

<mapping class="com.example.Library" />

<mapping class="com.example.Book" />

</session-factory>

</hibernate-configuration>

Now, let's implement the CRUD operations:

**Create (Insert) Library and Books:**

public class Main {

public static void main(String[] args) {

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Library.class)

.addAnnotatedClass(Book.class)

.buildSessionFactory();

Session session = factory.getCurrentSession();

try {

session.beginTransaction();

Library library = new Library("Main Library");

Book book1 = new Book("Book 1", library);

Book book2 = new Book("Book 2", library);

library.addBook(book1);

library.addBook(book2);

session.save(library);

session.getTransaction().commit();

} finally {

factory.close();

}

}

}

**Read (Retrieve) Library and Books:**

public class Main {

public static void main(String[] args) {

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Library.class)

.addAnnotatedClass(Book.class)

.buildSessionFactory();

Session session = factory.getCurrentSession();

try {

session.beginTransaction();

Library library = session.get(Library.class, 1); // Replace 1 with the library's ID to retrieve

List<Book> books = library.getBooks();

for (Book book : books) {

System.out.println("Book Title: " + book.getTitle());

}

session.getTransaction().commit();

} finally {

factory.close();

}

}

}

**Update Library and Books:**

public class Main {

public static void main(String[] args) {

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Library.class)

.addAnnotatedClass(Book.class)

.buildSessionFactory();

Session session = factory.getCurrentSession();

try {

session.beginTransaction();

Library library = session.get(Library.class, 1); // Replace 1 with the library's ID to update

library.setName("Updated Library Name");

session.getTransaction().commit();

} finally {

factory.close();

}

}

}

**Delete Library and Books:**

public class Main {

public static void main(String[] args) {

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Library.class)

.addAnnotatedClass(Book.class)

.buildSessionFactory();

Session session = factory.getCurrentSession();

try {

session.beginTransaction();

Library library = session.get(Library.class, 1); // Replace 1 with the library's ID to delete

session.delete(library);

session.getTransaction().commit();

} finally {

factory.close();

}

}

}

This exercise covers the basics of working with a one-to-many relationship in Hibernate for a library management system. Make sure to adapt the code and configurations to your specific requirements and update the Hibernate configuration accordingly.