**Entity Mapping and CRUD Operations:**

Discuss the process of mapping entities and performing CRUD operations using Spring ORM with Hibernate. Provide examples of annotating Java classes (@Entity, @Table, @Id, @Column) and mapping them to database tables. Demonstrate how to use Hibernate SessionFactory and HibernateTemplate (or JpaRepository) to perform basic CRUD operations (save, findById, update, delete).

**Entity Mapping**

To map Java classes to database tables, you need to annotate the classes with Hibernate-specific annotations.

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private Long id;

@Column(name = "username")

private String username;

@Column(name = "email")

private String email;

// Getters and setters

}

In this example:

@Entity indicates that the User class is an entity that can be mapped to a database table.

@Table specifies the name of the database table that the entity is mapped to.

@Id specifies the primary key of the entity.

@GeneratedValue specifies the strategy for generating the primary key value.

@Column specifies the column name and other attributes for each field.

CRUD Operations using Hibernate SessionFactory

To perform CRUD operations, you need to use the Hibernate SessionFactory to create a Session object.

@Repository

public class UserRepository {

@Autowired

private SessionFactory sessionFactory;

public void save(User user) {

Session session = sessionFactory.getCurrentSession();

session.beginTransaction();

session.save(user);

session.getTransaction().commit();

}

public User findById(Long id) {

Session session = sessionFactory.getCurrentSession();

session.beginTransaction();

User user = session.get(User.class, id);

session.getTransaction().commit();

return user;

}

public void update(User user) {

Session session = sessionFactory.getCurrentSession();

session.beginTransaction();

session.update(user);

session.getTransaction().commit();

}

public void delete(Long id) {

Session session = sessionFactory.getCurrentSession();

session.beginTransaction();

User user = session.get(User.class, id);

session.delete(user);

session.getTransaction().commit();

}

}

In this example, we use the SessionFactory to create a Session object, which is used to perform CRUD operations.

**CRUD Operations using HibernateTemplate**

Alternatively, you can use the HibernateTemplate to simplify CRUD operations.

@Repository

public class UserRepository {

@Autowired

private HibernateTemplate hibernateTemplate;

public void save(User user) {

hibernateTemplate.save(user);

}

public User findById(Long id) {

return hibernateTemplate.get(User.class, id);

}

public void update(User user) {

hibernateTemplate.update(user);

}

public void delete(Long id) {

User user = hibernateTemplate.get(User.class, id);

hibernateTemplate.delete(user);

}

}

**CRUD Operations using JpaRepository**

If you're using Spring Data JPA, you can use the JpaRepository interface to perform CRUD operations.

public interface UserRepository extends JpaRepository<User, Long> {

// No implementation needed

}