**Extra Questions**

1. **Sessions in Hibernate**

Hibernate session basically establishes a connection between the application and the persistence DB layer in order to perform CRUD operations on the entity using Hibernate. Hibernate sessions manages the life cycle of the persistent objects. Sessions are also used to manage the transactions.

For instance, hibernate session has beginTransaction() method. And after performing the operations, the transaction can be committed and on exception (or error) rollback the transaction. Sessions in hibernate also act as first level cache with only one copy of the object in the session. The changes made due to CRUD operations in the session are synchronized (persisted into the database) with the database when the session is flushed.

1. **Transaction Management in Spring and Hibernate**

Spring transaction management is handled using Spring AOP. In spring, we use the @Transactional annotation or XML configuration to do the transaction management. Spring framework provides abstraction to the transaction management.

The transaction management can also be handled programmatically to achieve the ACID properties.

A new transaction instance with appropriate transaction attributes s requested from the spring transaction manager by calling and based on the transactions success or failure, the transaction can be committed or rolled back.

In Hibernate, the transaction interface provides certain methods to handle the transaction management. A transaction is associated with a hibernate session. Similar to the Spring transaction management, a hibernate transaction is begun and all the operations are performed. When everything is successful, the transaction can be committed and when an exception occurs roll back the transaction.

**Pseudo code:**

|  |
| --- |
| 1. Session session = null; 2. Transaction tx = null; 4. try { 5. session = sessionFactory.openSession(); 6. tx = session.beginTransaction(); 7. // perform some actions here (such as DB creates/updates) 9. tx.commit(); 11. }catch (Exception ex) { 12. ex.printStackTrace(); 13. Log.error( <<error message>>); 14. //roll back when an exception occurs 15. tx.rollback(); 16. } 17. finally {session.close();} |

One small real-time example from one of my projects is as below:

When a refund is requested on a Purchase Order(PO) , for instance, then the Subscription should be Expired and the PO state should be refunded. All these operations should present in one single transaction. If any of these operations fail to write (update) to the DB, then the entire transaction should get rolled back.

BeginTransaction:

Update Subscription to Expired

Update PO to Refund

CommitTransaction

Exception:

RollbackTransaction.