**What is Session in Hibernate?**

Solution:- Session in one of the main runtime interfaces which acts as a bridge between Java application and Hibernate. Session is used to get a physical connection to the database. Session objects are not thread safe and should not be kept open for long time and so they are created and destroyed as needed. Session objects are lightweight hence they are instantiated during each interaction with the database as they are not thread safe. Sessions can be used to save, retrieve, delete or update object model to a database.

**How would you control transactions with Spring and Hibernate?**

Solution:- Transaction takes a database from one consistent state to another. Transaction is best described or should follow the following four properties. 1) Atomicity- A transaction consists of many steps, if all the steps in transaction are completed it is reflected in database or if any step fails it rolls back. 2) Consistency- database moves from one consistent state to another if transaction succeeds or remains in original state. 3) Isolation- Every transaction operates as if it is only transaction in system. 4) Durability- once transaction completes, updated rows must be available for all transactions permanently.

Spring framework provides an abstraction layer on top of different underlying transaction management API’s. Spring JPA provides JPA implementation of Hibernate.

Transaction management in spring and hibernate can be done in following way.

Transaction is a unit of work and it is associated with Session. Transaction interface provides following methods to control transaction.

1. **void begin()** used to start a new transaction.
2. **void commit()** it is used to end a transaction i.e. a unit of work.
3. **void rollback()** if any exception occurs a transaction is rolled back, so that the resources are free.
4. **void setTimeout(int seconds)** it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
5. **boolean isAlive()** checks if the transaction is still alive.
6. **void registerSynchronization(Synchronization s)** registers a user synchronization callback for this transaction.
7. **boolean wasCommited()** checks if the transaction is commited successfully.
8. **boolean wasRolledBack()** checks if the transaction is rolledback successfully.