✅ Top 20 Spring Boot Transaction Interview Questions with Detailed Answers

---

1. What is @Transactional in Spring?
`@Transactional` is an annotation used in Spring to manage transaction boundaries. It ensures that the annotated method is executed within a database transaction. If an exception occurs, the transaction is rolled back.

---

2. What is the default propagation behavior of @Transactional?
The default propagation is `Propagation.REQUIRED`. This means if a transaction already exists, the method will join it; otherwise, a new transaction will be started.

---

3. What are the different propagation types in Spring?

- `REQUIRED` – Uses the existing transaction or starts a new one. *(Default)*
- `REQUIRES_NEW` – Suspends the existing transaction and starts a new one.
- `NESTED` – Executes within a nested transaction.
- `SUPPORTS` – Joins the current transaction if available.
- `NOT_SUPPORTED` – Suspends any transaction and executes non-transactionally.
- `MANDATORY` – Must run within an existing transaction.
- `NEVER` – Must run without a transaction.

---

4. How does rollback work in Spring @Transactional?
By default, Spring rolls back a transaction **only for unchecked exceptions** (i.e., subclasses of `RuntimeException` and `Error`). You can override this using the `rollbackFor` attribute.

```
@Transactional(rollbackFor = Exception.class)
```

---

5. What is the isolation level in Spring transactions?
Isolation defines how transaction integrity is visible to other transactions. Spring supports:

- `DEFAULT`
- `READ_UNCOMMITTED`
- `READ_COMMITTED`
- `REPEATABLE_READ`
- `SERIALIZABLE`

These correspond to standard SQL isolation levels.

---

6. What is the default isolation level in Spring?
`Isolation.DEFAULT`, which uses the default isolation level of the underlying database (commonly `READ_COMMITTED` in most databases).

---

7. Can we apply @Transactional at the class level?
Yes. When applied at the class level, all public methods within the class are transactional unless overridden at the method level.

---

8. Can we use @Transactional on private methods?
No. Spring uses proxies to implement transactions, so `@Transactional` **only works on public methods**.

---

9. What is the difference between checked and unchecked exceptions in transaction management?

- **Unchecked (RuntimeException)**: Rollback by default.
- **Checked (Exception)**: Not rolled back by default unless specified using `rollbackFor`.

---

10. What is the use of @EnableTransactionManagement?
This annotation enables Spring's annotation-driven transaction management capability. It's typically added to a configuration class.

```
@EnableTransactionManagement
@Configuration
public class AppConfig { }
```

---

11. How does Spring manage transactions internally?
Spring uses **AOP (Aspect-Oriented Programming)** to create proxies around transactional methods and manage transaction boundaries (begin, commit, rollback).

---

12. What is PlatformTransactionManager?
It is the central interface in Spring's transaction infrastructure. Implementations include:

- `DataSourceTransactionManager`
- `JpaTransactionManager`
- `HibernateTransactionManager`

---

13. What happens when one method annotated with @Transactional calls another in the same class?

The internal method call **bypasses the proxy**, so transaction management **does not apply**. This is called the **self-invocation** issue.

---

14. What is the difference between @Transactional(propagation=REQUIRES_NEW) and REQUIRED?

- `REQUIRED`: Joins existing transaction or starts a new one.
- `REQUIRES_NEW`: Suspends any existing transaction and always starts a new one.

---

15. How can you test transactional behavior in a Spring Boot test?
Use `@Transactional` on test methods to rollback database changes after test execution.

```
@SpringBootTest
@Transactional
public class MyServiceTest { ... }
```

---

16. What is the use of TransactionTemplate in Spring?
`TransactionTemplate` provides a programmatic way to handle transactions, useful when you need fine-grained control.

```
transactionTemplate.execute(status -> {
    // your code
    return result;
});
```

---

17. Can we use @Transactional in REST controllers?
Yes, but it's not recommended for complex transaction logic. Services should handle transactions, and controllers should only delegate.

---

18. What is @TransactionalEventListener?
It allows listening to events within a transaction lifecycle. You can configure it to run after transaction commit or rollback.

```
@TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
public void handleEvent(MyEvent event) {
    // Logic after transaction commits
}
```

---

19. What is the difference between @Transactional and @Modifying in Spring Data JPA?

- `@Transactional`: Defines transaction boundaries.
- `@Modifying`: Used on repository methods to indicate a modifying query (e.g., update/delete). Often used **together** with `@Transactional`.

---

20. What is the use of propagation NESTED?
It creates a **savepoint** within the existing transaction. If an exception occurs, only the nested part rolls back, not the entire transaction.

Bonus :

Watch Spring Transactions Playlist For Detailed Understanding

Spring Transactions

https://www.youtube.com/playlist?list=PL-bgVzzRdaPimI4ERQ9gOtUKLEIALmoFL