**🔹 1. What is the difference between JPA and Hibernate?**

| **Feature** | **JPA** | **Hibernate** |
| --- | --- | --- |
| Type | Specification (interface) | Implementation |
| Dependency | javax.persistence | org.hibernate |
| Vendor | Multiple implementations | One of JPA's implementations |
| Use | Annotations and API | Adds extra features (caching, filters) |

➡️ **Hibernate implements JPA**. JPA is a standard; Hibernate is one implementation of that standard.

**🔹 2. How does Spring Boot integrate JPA and Hibernate?**

* Spring Boot uses **spring-boot-starter-data-jpa**, which internally:
  + Auto-configures **Hibernate** as JPA provider.
  + Manages **EntityManagerFactory**, **DataSource**, **TransactionManager**.
  + Enables **CRUDRepository**, **JpaRepository**, etc.

**🔹 3. What are the key annotations used in JPA?**

| **Annotation** | **Description** |
| --- | --- |
| @Entity | Marks class as a JPA entity |
| @Table | Maps entity to a database table |
| @Id | Primary key |
| @GeneratedValue | Auto-generate ID |
| @OneToOne, @OneToMany, @ManyToOne, @ManyToMany | For relationships |
| @JoinColumn | Specifies foreign key column |
| @Transient | Skip mapping field to DB column |
| @NamedQuery | Predefined JPQL queries |

**🔹 4. What is the difference between EntityManager and Session?**

| **Aspect** | **EntityManager (JPA)** | **Session (Hibernate)** |
| --- | --- | --- |
| Standard | JPA | Hibernate |
| Flush control | Yes | Yes |
| Cache interaction | 1st-level only | 1st and 2nd-level supported |
| Extra features | Limited | Filters, Interceptors, etc. |

**🔹 5. What is the difference between save() and persist() in JPA?**

* save() (Hibernate specific):
  + Returns the generated ID.
  + Can be used outside of transactions.
* persist() (JPA):
  + Returns void.
  + Should be used in a transaction.
  + Throws exception if entity already exists.

**🔹 6. What is the lifecycle of a JPA entity?**

1. **New/Transient** – not associated with persistence context.
2. **Managed** – attached to persistence context.
3. **Detached** – previously managed, now removed from context.
4. **Removed** – marked for deletion.

**🔹 7. What are different strategies of @GeneratedValue?**

java

CopyEdit

@GeneratedValue(strategy = GenerationType.IDENTITY) // DB auto increment

@GeneratedValue(strategy = GenerationType.SEQUENCE) // DB sequence

@GeneratedValue(strategy = GenerationType.TABLE) // Table-based sequence

@GeneratedValue(strategy = GenerationType.AUTO) // Pick default (DB-specific)

**🔹 8. How does @Transactional work in Spring Data JPA?**

* Begins a transaction before method execution.
* Commits transaction if no exception.
* Rolls back on unchecked exceptions (RuntimeException).
* Can be fine-tuned using rollbackFor, noRollbackFor, etc.

**🔹 9. What is Lazy vs Eager Fetching?**

* **EAGER**: Loads related entities immediately (can cause N+1 issues).
* **LAZY**: Loads only when needed (proxy object used).

**Example:**

java

CopyEdit

@OneToMany(fetch = FetchType.LAZY)

private List<Order> orders;

**🔹 10. What is N+1 problem in JPA and how to fix it?**

**Problem:** When loading one entity, then lazily loading many related entities (N additional queries).

**Fixes:**

* Use @EntityGraph
* Use JOIN FETCH in JPQL
* Batch fetch (hibernate.default\_batch\_fetch\_size)

**🔹 11. How do you optimize performance in JPA/Hibernate?**

* Use **second-level cache** (EhCache, Redis).
* Use **batch fetching**.
* Avoid **N+1 problems** with JOIN FETCH.
* Limit data with **pagination** (Pageable).
* Use **DTO projections** instead of entities for read-heavy APIs.

**🔹 12. What is dirty checking in Hibernate?**

* Hibernate automatically tracks changes in entities and updates the DB only if values have changed.
* It happens during flush().

**🔹 13. How do you handle optimistic locking?**

java

CopyEdit

@Version

private Integer version;

* Prevents lost updates in concurrent transactions.
* Throws OptimisticLockException if versions mismatch.

**🔹 14. Explain the role of JpaRepository, CrudRepository, and PagingAndSortingRepository.**

| **Interface** | **Features** |
| --- | --- |
| CrudRepository | Basic CRUD operations |
| PagingAndSortingRepository | Adds pagination & sorting |
| JpaRepository | Adds JPA features like flush(), batch updates, etc. |

**🔹 15. What is the use of @Modifying and @Query?**

java

CopyEdit

@Modifying

@Query("UPDATE User u SET u.name = :name WHERE u.id = :id")

int updateUserName(@Param("id") Long id, @Param("name") String name);

* Used for **DML operations** (UPDATE/DELETE).
* Must be marked with @Modifying.

**🔹 16. How do you perform auditing in Spring Data JPA?**

Enable JPA Auditing:

java

CopyEdit

@EnableJpaAuditing

public class AppConfig {}

Add fields:

java

CopyEdit

@CreatedDate

@LastModifiedDate

@CreatedBy

@LastModifiedBy

Implement AuditorAware for current user.

**🔹 17. What are common exceptions in JPA/Hibernate?**

* LazyInitializationException
* EntityNotFoundException
* OptimisticLockException
* NonUniqueResultException
* PersistenceException

**🔹 18. How do you use DTO projection in Spring Data JPA?**

java

CopyEdit

public interface UserDTO {

String getName();

String getEmail();

}

@Query("SELECT u.name AS name, u.email AS email FROM User u")

List<UserDTO> findAllUsersProjected();

**🔹 19. What is cascade in JPA?**

* Used to propagate operations from parent to child:

java

CopyEdit

@OneToMany(cascade = CascadeType.ALL)

private List<Order> orders;

**Types:** ALL, PERSIST, MERGE, REMOVE, REFRESH, DETACH

**🔹 20. How to implement soft delete in JPA?**

1. Add column:

java

CopyEdit

@Column

private Boolean deleted = false;

1. Override queries with deleted=false.
2. Use @Where(clause = "deleted = false") (Hibernate-specific).

If you want all of these in **PDF format**, or need **coding exercises**, case studies or **real-time scenarios**, just let me know!

**You said:**

what are joneral exception comes with above topics ?

**ChatGPT said:**

Here’s a comprehensive list of **common exceptions** that arise while working with **Spring Boot, JPA, and Hibernate**, grouped by category and **mapped to real scenarios** (especially relevant for experienced developers like you):

**✅ General JPA / Hibernate Exceptions**

| **Exception** | **Cause / Scenario** | **Notes** |
| --- | --- | --- |
| javax.persistence.EntityNotFoundException | Accessing an entity that doesn't exist using getOne() or getReference() | Use findById() to avoid lazy proxy exceptions. |
| javax.persistence.PersistenceException | Base class for all persistence exceptions | Wraps lower-level Hibernate errors. |
| javax.persistence.NonUniqueResultException | When a JPQL query expected single result but got more | Use getResultList() instead of getSingleResult(). |
| javax.persistence.NoResultException | getSingleResult() returned nothing | Avoid if possible—check first or use Optional. |
| javax.persistence.OptimisticLockException | Version mismatch in optimistic locking | Ensure @Version column is used properly. |
| javax.persistence.TransactionRequiredException | Operation needs a transaction but none exists | Happens when modifying entities without @Transactional. |

**✅ Hibernate-Specific Exceptions**

| **Exception** | **Cause / Scenario** | **Notes** |
| --- | --- | --- |
| org.hibernate.LazyInitializationException | Accessing lazy-loaded data **after session is closed** | Ensure fetch happens within the transaction or use DTO projection. |
| org.hibernate.ObjectNotFoundException | Lazy proxy cannot initialize object | Usually with getReference() when entity doesn’t exist. |
| org.hibernate.StaleObjectStateException | Data updated/removed by another transaction | Usually related to optimistic locking. |
| org.hibernate.PropertyValueException | Saving an entity with null in non-nullable column | Check DB constraints and validation. |
| org.hibernate.QueryException | Invalid HQL syntax | Validate JPQL or HQL during testing. |
| org.hibernate.HibernateException | Top-level unchecked Hibernate exception | Often wraps SQL or mapping issues. |

**✅ Spring Data JPA Specific Exceptions**

| **Exception** | **Cause / Scenario** | **Notes** |
| --- | --- | --- |
| org.springframework.dao.EmptyResultDataAccessException | Deleting non-existent entity | repository.deleteById(id) where id doesn’t exist. |
| org.springframework.dao.IncorrectResultSizeDataAccessException | Query returns wrong number of results | Fix query logic or adjust expectations. |
| org.springframework.dao.DataIntegrityViolationException | DB constraint violated (e.g., FK, unique) | Maps to JDBC integrity constraint violations. |
| org.springframework.dao.InvalidDataAccessApiUsageException | Using repository incorrectly, e.g. null ID | Common with misuse of CrudRepository. |
| org.springframework.orm.jpa.JpaObjectRetrievalFailureException | Underlying entity not found | Usually a cascading issue or wrong relationship. |
| org.springframework.transaction.TransactionSystemException | Transaction failed (e.g., validation) | Wraps exceptions inside a transactional context. |

**✅ Transaction & EntityManager Issues**

| **Exception** | **Cause / Scenario** | **Notes** |
| --- | --- | --- |
| javax.persistence.RollbackException | Commit failed due to constraint or exception | Check inner cause: validation, constraint, etc. |
| org.springframework.transaction.UnexpectedRollbackException | Exception occurred inside a transaction marked for rollback | Any runtime exception triggers rollback unless configured otherwise. |
| IllegalStateException | Operating on closed EntityManager | Happens when accessing outside transactional boundary. |

**✅ Validation / Constraint Exceptions**

| **Exception** | **Cause / Scenario** | **Notes** |
| --- | --- | --- |
| javax.validation.ConstraintViolationException | Bean validation fails (@NotNull, @Size, etc.) | Usually thrown when saving/updating entities. |
| org.hibernate.exception.ConstraintViolationException | DB-level violation (unique, FK) | Often wrapped in DataIntegrityViolationException. |

**✅ Query & Mapping Issues**

| **Exception** | **Cause / Scenario** | **Notes** |
| --- | --- | --- |
| IllegalArgumentException | JPQL with incorrect params or entity not managed | E.g., using @Query with wrong parameter names. |
| MappingException | Invalid or duplicate entity mapping | E.g., two classes mapping to the same table. |
| AnnotationException | Incorrect annotations (e.g., missing @Id) | Ensure all entities are properly annotated. |

**🔁 How to Handle These Gracefully in Production:**

* Use @ControllerAdvice and @ExceptionHandler to map exceptions to friendly responses.
* Avoid getOne() or getReference() if you're not sure the entity exists.
* Use Optional in repository methods.
* Enable detailed logs for Hibernate SQL and transactions:

yaml

CopyEdit

spring.jpa.show-sql: true

spring.jpa.properties.hibernate.format\_sql: true

logging.level.org.hibernate.SQL: DEBUG

logging.level.org.hibernate.type.descriptor.sql.BasicBinder: TRACE