**Hibernate Interview Questions**

**1**. What is Hibernate Framework?

* Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.
* Hibernate is one of the best ORM framework available for java.

**2.** What is ORM Hibernate?

* Hibernate ORM stands for (object relational mapping).
* This tool is mainly used for converting data in RDBMS to an object used in object-oriented programming constructs.
* It greatly helps in simplifying data retrieval, creation and manipulation.

3. What are some of the important interfaces of Hibernate framework?

* Hibernate core interfaces are:

1. Configuration
2. SessionFactory
3. Session
4. Criteria
5. Query
6. Transaction

**4.** What is configuration?

* The purpose of configuration interface is to configure hibernate settings and mappings.

**Example**:

Configuration configuration = new Configuration ();

configuration.configure("hibernate.cfg.xml"); // Load configuration from XML file

SessionFactory sessionFactory = configuration.buildSessionFactory();

* The configuration interface serves as the starting point for setting up hibernate in your application
* It plays a vital role in defining how your Java objects are persisted and retrieved from the database.

**5**. What is transaction?

* In the context of Hibernate and database management, a transaction refers to a sequence of one or more database operations that are executed as a single, atomic unit of work.
* The purpose of a transaction is to ensure that a series of database operations either all succeed (commit) or all fail (rollback).

**6.** What is Session and Session Factory in Hibernate?

* The Session Factory in Hibernate is responsible for creating of session objects.
* The Hibernate Session provides methods such as Update, Delete, Post, Get all, all the Crud based operations on database to which the session factory is connected.

**7.** What is caching in hibernate?

* Caching is a mechanism to enhance the performance of an application.
* Caching is used to reduce the number of database queries.

**8.** What is Hibernate first level and second level cache?

* The first level caching is only associated with session level and accessible only at the session level.
* Whereas second level caching is associated with session factory and it is available to all the sessions.

**9.** What is the benefit of using Hibernate?

Some of the important advantages of Hibernate framework over JDBC are:

* Hibernate removes a lot of boiler-plate code that comes with JDBC API, the code looks cleaner and more readable.
* Hibernate supports inheritance, associations and collections. These features are not present with JDBC API.

//@ManyToMany(mappedBy = "admins", cascade = CascadeType.ALL)

@ManyToMany( cascade = CascadeType.ALL,fetch = FetchType.LAZY)

@JoinTable(name = "property\_admin", joinColumns = {

@JoinColumn(name = "property\_id") }, inverseJoinColumns = {

@JoinColumn(name = "admin\_id") })

//@Column(nullable = true)

//@JsonIgnore

@JsonBackReference

private List<Property> properties;

**10.** What is cascade and fetch?

* In the code snippet you provided, cascade and fetch are annotations used in the context of JPA (Java Persistence API) to define the behaviour of a many-to-many relationship between two entities.

**cascade** = CascadeType.ALL: The cascade attribute specifies how operations (such as persist, merge, remove, etc.) should be cascaded from one entity to another. In this case, CascadeType.ALL means that all operations performed on the owning entity (the one where this annotation is defined) will be cascaded to the associated entities. So, if an operation like persist or remove is performed on the owning entity, the same operation will be applied to the associated entities.

CascadeType.ALL: This option cascades all operations (persist, merge, remove, refresh) from the parent entity to the associated entities. Any change in the parent entity will be propagated to the associated entities.

CascadeType.PERSIST: This option cascades the persist operation from the parent entity to the associated entities. When you save the parent entity, the associated entities will also be saved if they are not already persistent.

CascadeType.MERGE: This option cascades the merge operation from the parent entity to the associated entities. Changes to the parent entity will be merged into the associated entities when you merge the parent entity.

CascadeType.REMOVE: This option cascades the remove operation from the parent entity to the associated entities. When you delete the parent entity, the associated entities will also be deleted.

CascadeType.REFRESH: This option cascades the refresh operation from the parent entity to the associated entities. It refreshes the state of the associated entities with the current state from the database.

**fetch** = FetchType.LAZY: The fetch attribute determines how the associated entities should be loaded from the database. In this case, FetchType.LAZY indicates that the associated entities should be loaded lazily, meaning they will be fetched from the database only when explicitly accessed. This is useful when you have large collections of associated entities, and you want to avoid fetching them all at once, which could impact performance.

Regarding the other annotations in your code snippet:

**@JoinTable**: Specifies the details of the join table used to store the many-to-many relationship between the entities.

**@JoinColumn**: Specifies the column used for joining the owning entity with the associated entity in the join table.

**@JsonBackReference** (from Jackson library): Indicates that the annotated property should be ignored during JSON serialization to prevent circular references when serializing bidirectional relationships.