Hibernate & it's annotations

Trainer Name:
Shreyansh

Introduction to Hibernate

Hibernate is an open-source ORM (Object-Relational Mapping) framework for Java that simplifies database interactions by mapping Java objects to database tables. It eliminates the need for boilerplate JDBC code and provides advanced features like caching, lazy loading, and transaction management.

Core Features of Hibernate

- **ORM (Object-Relational Mapping)**: Maps Java objects to relational database tables.
- **Hibernate Query Language (HQL)**: A powerful query language similar to SQL but works with entity objects.
- **Automatic Table Creation**: Generates tables automatically based on entity classes.
- **Transaction Management**: Handles database transactions efficiently.
- **Caching Mechanism**: Improves performance by reducing database access.
- **Lazy and Eager Loading**: Controls how data is fetched from the database.

Hibernate Architecture

Components:

1. **SessionFactory**: Responsible for creating Session objects.
2. **Session**: Provides an interface to perform CRUD operations.
3. **Transaction**: Manages transactions at the application level.
4. **Query and Criteria API**: Enables querying the database.
5. **Configuration**: Loads settings from hibernate.cfg.xml or hibernate.properties.

Diagram:

Application

|

SessionFactory

|

Session

|

Transaction

|

Database

Hibernate Configuration

Hibernate can be configured using an XML file (hibernate.cfg.xml) or programmatically.

hibernate.cfg.xml Example:

```
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mydb</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">password</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
  </session-factory>
</hibernate-configuration>
```

Hibernate Annotations

Commonly Used Annotations in Hibernate

Annotation	Description
@Entity	Marks a class as a Hibernate entity.
@Table(name="table_name")	Specifies the database table name.
@Id	Marks the primary key field.
@GeneratedValue(strategy=GenerationType.IDENTITY)	Auto-generates primary key values.
@Column(name="column_name")	Maps a field to a database column.
@OneToOne, @OneToMany, @ManyToOne, @ManyToMany	Defines relationships between entities.
@JoinColumn(name="column_name")	Specifies the foreign key column.
@Transient	Excludes a field from persistence.

Example:

```
@Entity
@Table(name="employees")
public class Employee {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @Column(name="name", nullable=false)
  private String name;
```

```
    @OneToOne
    @JoinColumn(name="address_id")
    private Address address;
}
```

Hibernate Caching

Caching improves performance by reducing database calls.

First-Level Cache

- Enabled by default.
- Works at the Session level.
- Cache is cleared when the session is closed.

Example:

```
Session session = sessionFactory.openSession();
Employee emp1 = session.get(Employee.class, 1); // Fetches from DB
Employee emp2 = session.get(Employee.class, 1); // Fetches from cache
```

Second-Level Cache

- Works at the SessionFactory level.
- Requires explicit configuration (e.g., EHCache, Redis, Hazelcast).

Configuration for EHCache:

```
<property name="hibernate.cache.use_second_level_cache">true</property>
<property name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>
```

Annotation Example:

```
@Entity
@Cacheable
```

```
@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Employee {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String name;
}
```

Hibernate Relationships

One-to-One Relationship

```
@Entity
public class Employee {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @OneToOne
  @JoinColumn(name="address_id")
  private Address address;
}
```

One-to-Many Relationship

```
@Entity
public class Department {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;

    @OneToMany(mappedBy = "department", cascade = CascadeType.ALL)
    private List<Employee> employees;
}
```

Many-to-Many Relationship

```
@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToMany
    @JoinTable(name="student_course",
            joinColumns=@JoinColumn(name="student_id"),
            inverseJoinColumns=@JoinColumn(name="course_id"))
    private List<Course> courses;
}
```

Hibernate Query Language (HQL)

HQL is an object-oriented query language similar to SQL but works with entity objects instead of tables.

Basic HQL Query

```
Query query = session.createQuery("from Employee");
List<Employee> employees = query.list();
```

HQL with Conditions

```
Query query = session.createQuery("from Employee where id = :id");

query.setParameter("id", 1);

Employee employee = (Employee) query.uniqueResult();
```

Conclusion

Hibernate is a powerful ORM framework that simplifies Java database interactions with features like annotations, caching, relationships, and HQL. Understanding these concepts enables efficient development of scalable applications.

Hibernate interview Questions

Basic Questions

1. What is Hibernate?
   Hibernate is an open-source Object-Relational Mapping (ORM) framework for Java. It simplifies database interactions by allowing developers to work with Java objects rather than SQL queries. Hibernate automates CRUD operations and manages relationships between entities.

2. Explain the advantages of using Hibernate over JDBC.
   - Reduces boilerplate code.
   - Supports object-oriented programming.
   - Provides caching mechanisms to improve performance.
   - Offers HQL (Hibernate Query Language) for database independence.
   - Supports automatic transaction management.

3. What are the core components of Hibernate?
   - SessionFactory: Provides session objects for database operations.
   - Session: Represents a single unit of work with the database.
   - Transaction: Manages transactions.
   - Query: Used to retrieve objects from the database.
   - Configuration: Reads and loads configuration settings.

4. What is an ORM (Object-Relational Mapping)?
   ORM is a technique that converts data between a relational database and object-oriented programming languages. It eliminates the need for manual SQL queries.

5. What are the key features of Hibernate?
   - ORM support
   - HQL (Hibernate Query Language)
   - Caching support
   - Lazy loading
   - Automatic primary key generation
   - Annotations and XML-based configuration

6. What is the Hibernate architecture?
   Hibernate architecture consists of:
   - Configuration
   - SessionFactory
   - Session
   - Transaction
   - Query and Criteria API
   - Cache Mechanism

7. Explain Hibernate SessionFactory.
   SessionFactory is a factory for session objects. It is a heavyweight object and should be created once per application.

8. What is the difference between Session and SessionFactory?
   - SessionFactory: Creates session objects and is a heavyweight object.
   - Session: Represents a single unit of work and is a lightweight object.

9. What are the different states of a Hibernate object?
   - Transient: Object is not associated with Hibernate.
   - Persistent: Object is associated with a session.
   - Detached: Object is no longer associated with a session.

10. What is a Hibernate configuration file?
    It is an XML file (hibernate.cfg.xml) that contains database connection details, entity mappings, and Hibernate properties.

Mapping and Annotations

11. What is Hibernate Mapping?
    Hibernate Mapping defines how Java classes relate to database tables using XML or annotations.

12. Explain different types of Hibernate inheritance mapping.
    - Single Table Strategy (@Inheritance(strategy=InheritanceType.SINGLE_TABLE))
    - Table Per Class Strategy (@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS))
    - Joined Table Strategy (@Inheritance(strategy=InheritanceType.JOINED))

13. How do you define a primary key in Hibernate?
    Using @Id annotation or XML mapping.

14. What are the different types of Hibernate associations?
    - One-to-One
    - One-to-Many
    - Many-to-Many

15. How can we map a one-to-one relationship in Hibernate?
    Using @OneToOne annotation.

16. How can we map a one-to-many relationship in Hibernate?
    Using @OneToMany and @JoinColumn annotations.

17. How can we map a many-to-many relationship in Hibernate?
    Using @ManyToMany and @JoinTable annotations.

18. What is @Entity annotation in Hibernate?
    It marks a class as a Hibernate entity.

19. Explain @Table and @Column annotations in Hibernate.
    - @Table defines the database table.
    - @Column maps a field to a column.

20. What is the use of @JoinColumn in Hibernate?
    It specifies the foreign key column for associations.

Querying and Transactions

21. What are HQL and SQL in Hibernate?
    - HQL: Object-oriented query language.
    - SQL: Native SQL queries.
22. How does Criteria API work in Hibernate?
    It provides programmatic query creation.
23. What is Named Query in Hibernate?
    A pre-defined query using @NamedQuery annotation.
24. What is the difference between get() and load() methods?
    - get(): Returns null if not found.
    - load(): Throws an exception if not found.
25. How does Hibernate handle transactions?
    Using session.beginTransaction() and commit() methods.
26. What is the difference between commit() and flush() in Hibernate?
    - commit(): Ends a transaction.
    - flush(): Synchronizes session with the database.
27. Explain optimistic and pessimistic locking in Hibernate.
    - Optimistic Locking: Versioning mechanism.
    - Pessimistic Locking: Locks data in the database.
28. What is Lazy Loading and Eager Loading in Hibernate?
    - Lazy Loading: Data is loaded on demand.
    - Eager Loading: Data is loaded immediately.
29. How does Hibernate support caching?
    It provides first-level and second-level caching.
30. What are the types of caches in Hibernate?
    - First-Level Cache (Session Scope)
    - Second-Level Cache (SessionFactory Scope)

Performance and Best Practices

41. What are the common performance issues in Hibernate?
    - N+1 problem
    - Excessive joins
    - Inefficient queries
42. How can you optimize Hibernate performance?
    - Use caching
    - Optimize queries
    - Use batch processing
43. How does connection pooling work in Hibernate?
    It manages database connections efficiently.
44. What is batch processing in Hibernate?
    It processes multiple queries in one go.
45. What is the N+1 problem in Hibernate, and how do you solve it?
    - It occurs when separate queries are fired for each object.
    - Solved using JOIN FETCH or @BatchSize.
46. What are DTO and DAO patterns in Hibernate?
    - DTO (Data Transfer Object): Transfers data between layers.
    - DAO (Data Access Object): Encapsulates database operations.
47. How do you handle exceptions in Hibernate?
    Using try-catch blocks and HibernateException.
48. Can Hibernate work without a database?
    Yes, using an in-memory database like H2.
49. What is the role of FetchType in Hibernate?
    Defines loading strategy (LAZY/EAGER).
50. How do you implement soft delete in Hibernate?
    Using a deleted flag instead of physical deletion.

Diagrams:

1. Hibernate Architecture
2. Entity Relationships (One-to-One, One-to-Many, Many-to-Many)
3. Hibernate Caching Mechanism
4. Transaction Management in Hibernate
5. Hibernate Object States (Transient, Persistent, Detached)