Hibernate

---

1. Overview

Hibernate is a popular **Object-Relational Mapping (ORM)** framework for **Java** that simplifies database interactions by mapping Java objects to database tables.

- Released in 2001, widely used in enterprise Java applications.
- Provides a framework to map an object-oriented domain model to a traditional relational database.
- Helps avoid writing complex SQL by providing a high-level API for data access.
- Supports multiple databases and offers database independence.
- Works as a layer between Java application and the database.

---

2. Core Concepts

Object-Relational Mapping (ORM)

- Maps Java classes to database tables.
- Maps Java class attributes to table columns.
- Automatically generates SQL queries based on object state.

Session & SessionFactory

- **SessionFactory**: Thread-safe factory for **Session** objects. Heavyweight; usually one per application.
- **Session**: Represents a single unit of work with the database; used to create, read, update, and delete persistent objects.
- Session is **not thread-safe** and short-lived (typically one per transaction).

Persistent Objects & States

- **Transient**: New objects not associated with a session or database.
- **Persistent**: Objects associated with a session and saved in the database.

- **Detached**: Objects previously persistent but no longer associated with a session.

Transactions

- Hibernate supports transaction management.
- Integrates with JTA or JDBC transactions.
- Transactions ensure atomicity and consistency.

---

3. Mapping

Mapping Files or Annotations

- Entities are mapped using XML configuration files (.hbm.xml) or **Java annotations** (preferred).
- Annotations like @Entity, @Table, @Id, @Column define the mapping between classes and tables.

Example:

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

  @Column(name = "name")
  private String name;

  // getters and setters
}
```

Associations

- Relationships between entities:
    - **One-to-One** (@OneToOne)
    - **One-to-Many** (@OneToMany)
    - **Many-to-One** (@ManyToOne)
    - **Many-to-Many** (@ManyToMany)
- Support for cascading operations and lazy/eager fetching.

---

4. Querying

Hibernate Query Language (HQL)

- Object-oriented query language similar to SQL but operates on entities and properties.
- Supports polymorphic queries, joins, and projections.

Example:

String hql = "FROM Employee WHERE name = :employeeName";

Query query = session.createQuery(hql);

query.setParameter("employeeName", "John");

List results = query.list();

Criteria API

- Programmatic, type-safe API to build queries dynamically.
- Useful for complex queries without string concatenation.

Native SQL

- Supports direct SQL queries if needed.

---

5. Caching

- Improves performance by minimizing database hits.
- **First-level cache**: Session-level cache (mandatory, enabled by default).

- **Second-level cache**: Shared across sessions; optional and configurable (e.g., EHCache).
- Query cache: caches results of queries.

---

6. Configuration

- Hibernate configured via:
    - hibernate.cfg.xml file
    - Programmatic configuration
    - Properties file
- Config options include database connection, dialect, caching, show SQL, etc.

---

7. Integration

- Commonly used with **Spring Framework** for declarative transaction management and dependency injection.
- Supports JPA (Java Persistence API) specification, allowing Hibernate to be used as a JPA provider.

---

8. Advantages

- Eliminates boilerplate JDBC code.
- Database-independent, supporting multiple dialects.
- Powerful query capabilities (HQL, Criteria).
- Caching support improves performance.
- Easy transaction management.
- Mature, stable, and widely adopted.