🗄️ HIBERNATE HANDBOOK

(Interview + Practical – Complete Guide)

This handbook covers **all important Hibernate concepts** required for **interviews and real-world projects**. It is designed as a **one-stop revision guide**.

1️⃣ What is Hibernate?

Hibernate is an **ORM (Object Relational Mapping) framework** that maps Java objects to relational database tables.

Interview One-liner

> Hibernate is a Java ORM framework that simplifies database interaction by mapping objects to tables.

2️⃣ Hibernate vs JDBC

JDBC	Hibernate
Manual SQL	Automatic SQL
Boilerplate code	Minimal code
No caching	Built-in caching
Error-prone	Optimized & safe

3️⃣ Hibernate vs JPA

Hibernate	JPA
Implementation	Specification
Session API	EntityManager API
Hibernate annotations	Standard annotations

👉 Hibernate **implements JPA**.

---

4 Core Hibernate Architecture

◇ SessionFactory

- Heavyweight
- Created once
- Thread-safe

◇ Session

- Lightweight
- Represents DB connection
- Not thread-safe

◇ Transaction

- Ensures ACID properties

---

5 Hibernate Configuration

hibernate.cfg.xml

```
<hibernate-configuration>
 <session-factory>
  <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  <property name="hibernate.hbm2ddl.auto">update</property>
  <property name="hibernate.show_sql">true</property>
 </session-factory>
</hibernate-configuration>
```

---

6 Hibernate Annotations (MOST IMPORTANT)

- `@Entity`
- `@Table`
- `@Id`
- `@GeneratedValue`
- `@Column`
- `@Transient`

---

7 Entity Lifecycle States

State	Description

State	Description
Transient	New object
Persistent	Managed by Session
Detached	Session closed
Removed	Deleted

---

8 Storing Data in Hibernate (DETAILED)

◇ Complete Working Example (Parent–Child)

Employee.java

```
@Entity
@Table(name = "employee")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToMany(mappedBy = "employee", cascade = CascadeType.ALL)
    private List<Laptop> laptops;
}
```

Laptop.java

```
@Entity
@Table(name = "laptop")
public class Laptop {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String brand;

    @ManyToOne
    @JoinColumn(name = "emp_id")
    private Employee employee;
}
```

Storing Data (Main Class)

```
Session session = factory.openSession();
Transaction tx = session.beginTransaction();
```

```java
Employee emp = new Employee();
emp.setName("AJ");

Laptop l1 = new Laptop();
l1.setBrand("Dell");
l1.setEmployee(emp);

Laptop l2 = new Laptop();
l2.setBrand("HP");
l2.setEmployee(emp);

emp.setLaptops(List.of(l1, l2));

session.save(emp);   // cascades laptops

tx.commit();
session.close();
```

◇ What This Code Explains (Interview GOLD)

- Transient → Persistent → Detached
- Cascade save
- Owning side (`@ManyToOne`) stores FK
- Data saved only on commit

---

9 save() vs persist() vs saveOrUpdate()

Method	Usage
save()	Hibernate specific
persist()	JPA standard
saveOrUpdate()	Insert or update

---

10 Relationships Mapping

- `@OneToOne`
- `@OneToMany`
- `@ManyToOne`
- `@ManyToMany`

Owning Side

- Controls foreign key
- Usually `@ManyToOne`

1️⃣1️⃣ mappedBy

```
@OneToMany(mappedBy = "employee")
```

- Avoids extra column
- Defines inverse side

1️⃣2️⃣ Fetch Strategies

Fetch Type	Description
LAZY	Load on demand
EAGER	Load immediately

📌 Defaults:

- OneToMany → LAZY
- ManyToOne → EAGER

1️⃣3️⃣ Fetching Data in Hibernate (DETAILED)

◇ Basic Fetching

```
Session session = factory.openSession();
Employee emp = session.get(Employee.class, 1L);
System.out.println(emp.getName());
```

◇ LAZY Loading Example

```
System.out.println(emp.getLaptops().size()); // DB hit here
```

◇ LazyInitializationException Example

```
session.close();
emp.getLaptops().size(); // ❌ Exception
```

◇ JOIN FETCH (Recommended)

```
Session session2 = factory.openSession();
List<Employee> list = session2.createQuery(
    "SELECT e FROM Employee e JOIN FETCH e.laptops",
    Employee.class
).getResultList();
```

◇ N+1 Problem Example

```
List<Employee> emps = session2.createQuery("FROM Employee", Employee.class)
                              .getResultList();
```

```
for(Employee e : emps) {
    System.out.println(e.getLaptops().size()); // multiple queries
}
```

◇ Pagination Example

```
List<Employee> page = session2.createQuery("FROM Employee", Employee.class)
    .setFirstResult(0)
    .setMaxResults(5)
    .getResultList();
```

◇ get() vs load()

```
Employee e1 = session2.get(Employee.class, 1L);
Employee e2 = session2.load(Employee.class, 1L);
```

-————————— | ——————— | | Immediate DB hit | Proxy object | | Returns null | Throws exception |

◇ LazyInitializationException

Occurs when:

- LAZY data accessed after session close

```
session.close();
emp.getLaptops(); // ✕ Exception
```

◇ Solving LazyInitializationException

- Use `JOIN FETCH`
- Access data inside session
- Use DTO projection

◇ N+1 Query Problem

- 1 query for parent
- N queries for child collections

◇ Solution: JOIN FETCH

```
SELECT e FROM Employee e JOIN FETCH e.laptops
```

✓ Fetches data in single query

◇ Pagination (VERY IMPORTANT)

```
query.setFirstResult(0);
query.setMaxResults(10);
```

1️⃣8️⃣ Dirty Checking

Hibernate automatically detects changes and updates DB on commit.

Hibernate automatically detects changes and updates DB on commit.

1️⃣9️⃣ Caching in Hibernate

First-Level Cache

- Session-level
- Enabled by default

Second-Level Cache

- SessionFactory-level
- Needs configuration

2️⃣0️⃣ Second-Level Cache Setup (Overview)

- Enable cache properties
- Use Ehcache / Redis
- Mark entities cacheable

2️⃣1️⃣ Locking & Versioning

```
@Version
private int version;
```

- Optimistic locking

2️⃣2️⃣ Transactions & Rollback

```
try {
 tx.commit();
} catch(Exception e) {
 tx.rollback();
}
```

2️⃣3️⃣ Hibernate Exceptions

- ConstraintViolationException
- LazyInitializationException
- StaleObjectStateException

2️⃣4️⃣ Performance Best Practices

✓ Prefer LAZY fetching ✓ Use pagination ✓ Avoid EAGER collections ✓ Use JOIN FETCH ✓ Use DTO projection

---

🎯 HIBERNATE INTERVIEW CHECKLIST

✓ ORM & Hibernate basics ✓ Session vs SessionFactory ✓ Entity lifecycle ✓ save vs persist ✓ Relationships & owning side ✓ Fetch strategies ✓ LazyInitializationException ✓ HQL & JOIN FETCH ✓ N+1 problem ✓ Caching ✓ Locking ✓ Performance tuning

---

🏁 Final Advice

Strong Hibernate fundamentals automatically make JPA and Spring Data JPA easy.

📌 Next Steps:

- Spring Data JPA Handbook
- Hibernate interview questions
- Advanced performance tuning