

JPA HANDBOOK

(Interview + Practical – Complete Guide)

This handbook covers **everything you must know about JPA** from basics up to **interview readiness**, including how it relates to Hibernate and Spring Data JPA.

What is JPA?

JPA (Java Persistence API) is a **specification** that defines how Java objects are mapped to relational database tables.

👉 JPA is **not an implementation**. 👉 Hibernate is the **most popular implementation** of JPA.

Interview One-liner

JPA is a specification, Hibernate is an implementation, and Spring Data JPA is an abstraction on top of JPA.

JPA vs Hibernate

JPA	Hibernate
Specification	Implementation
Portable	Vendor specific
Uses <code>EntityManager</code>	Uses <code>Session</code>
<code>persist()</code>	<code>save()</code>

Core JPA Components

EntityManager

- Manages entity lifecycle
- Performs CRUD operations

EntityManagerFactory

- Creates EntityManager
- Heavyweight, created once

persistence.xml

- Configuration file for JPA
-

Basic JPA Configuration

File Path (VERY IMPORTANT)

```
src/main/resources/META-INF/persistence.xml
```

JPA will NOT work if this exact path is wrong.

Basic JPA Configuration

persistence.xml

```
<persistence xmlns="https://jakarta.ee/xml/ns/persistence" version="3.0">
  <persistence-unit name="myPU">
    <class>com.example.Employee</class>
    <properties>
      <property name="jakarta.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/testdb"/>
      <property name="jakarta.persistence.jdbc.user" value="root"/>
      <property name="jakarta.persistence.jdbc.password" value="root"/>
      <property name="jakarta.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibernate.show_sql" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

JPA Entity Mapping

@Entity

Marks a class as a persistent entity.

```
@Entity  
public class Employee { }
```

@Id & @GeneratedValue

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



JPA Entity Lifecycle States

State	Description
New	Not associated with EntityManager
Managed	Associated with persistence context
Detached	Session closed
Removed	Scheduled for deletion



Persisting Data in JPA

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("myPU");  
EntityManager em = emf.createEntityManager();  
EntityTransaction tx = em.getTransaction();  
  
tx.begin();  
Employee e = new Employee();  
e.setName("AJ");  
em.persist(e);  
tx.commit();
```

Important Notes

- `persist()` does **not return ID**
- Data is stored only on `commit()`

persist() vs merge()

persist	merge
New entity	Detached entity
No return	Returns managed entity
Insert only	Insert or Update

Relationships in JPA

@ManyToOne (Owning Side)

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

@OneToMany (Inverse Side)

```
@OneToMany(mappedBy = "employee")  
private List<Laptop> laptops;
```

 Foreign key is always managed by the **owning side**.

Fetch Strategies

LAZY

- Loads data when accessed
- Better performance

EAGER

- Loads immediately
- Can cause performance issues

 Defaults:

- `@ManyToOne` → EAGER
- `@OneToMany` → LAZY



JPQL (JPA Query Language)

```
SELECT e FROM Employee e WHERE e.name = :name
```



Uses **entity names**, not table names



A Complete JPA Program (Storing + Fetching + JPQL)



Project Structure

```
src/main/java/com/example
├── Employee.java
├── Laptop.java
└── MainApp.java

src/main/resources/META-INF
└── persistence.xml
```



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;
}

```

MainApp.java (Everything Explained)

```

EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("myPU");

EntityManager em = emf.createEntityManager();
EntityTransaction tx = em.getTransaction();

// ----- STORING DATA -----
tx.begin();

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));

em.persist(emp); // cascades laptops

tx.commit();

// ----- FETCHING DATA -----
Employee e = em.find(Employee.class, emp.getId());
System.out.println(e.getName());

```

```

// LAZY loading
System.out.println(e.getLaptops().size());

// ----- JPQL QUERY -----
TypedQuery<Employee> query = em.createQuery(
    "SELECT e FROM Employee e WHERE e.name = :name",
    Employee.class
);
query.setParameter("name", "AJ");

List<Employee> result = query.getResultList();

em.close();
emf.close();

```

What This Single Program Covers

👉 JPA configuration 👉 Entity mapping 👉 Relationships & owning side 👉 Cascade 👉 Persisting data 👉
Fetching data 👉 LAZY loading 👉 JPQL query 👉 EntityManager lifecycle

Fetch Join (Solving N+1)

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

👉 Fetches parent & child in a single query

Entity States Example

```

Employee e = new Employee(); // New
em.persist(e); // Managed
em.close(); // Detached

```

Transactions in JPA

- Mandatory for insert/update/delete
- Read-only operations may skip transaction

```
tx.begin();
// DB operations
tx.commit();
```

Locking & Versioning

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

👉 Prevents concurrent update conflicts

JPA with Spring Boot (Overview)

```
public interface EmployeeRepository
    extends JpaRepository<Employee, Long> { }
```

👉 No EntityManager code 👉 Auto CRUD methods

Common JPA Exceptions

- `EntityNotFoundException`
- `LazyInitializationException`
- `TransactionRequiredException`

JPA INTERVIEW CHECKLIST

👉 JPA vs Hibernate 👉 EntityManager lifecycle 👉 persist vs merge 👉 JPQL vs SQL 👉 Fetch strategies (LAZY/EAGER) 👉 Entity lifecycle states 👉 Relationships & owning side 👉 Transactions 👉 Locking & @Version 👉 Spring Data JPA basics

Final Advice

If you understand this handbook, you are **interview-ready for JPA** and can easily move to **Spring Data JPA**.

 Next Steps:

- Spring Data JPA handbook
- JPA interview questions & answers
- Real-time JPA problems & solutions