Hibernate Tutorial

1. Core Hibernate Concepts

Before touching code, understand the fundamentals:

- **What is Hibernate?**
  - An ORM (Object Relational Mapping) framework for Java that maps Java classes to database tables.
  - Eliminates boilerplate JDBC code.
  - Provides HQL (Hibernate Query Language) for object-based queries.
- **Key Features**
  - Automatic table creation (via `hbm2ddl.auto`).
  - First-level cache (Session cache).
  - Second-level cache (Ehcache, Redis, etc.).
  - Lazy loading & Eager loading.
  - Transaction management.
- **Important Interfaces/Classes**
  - `Configuration`
  - `SessionFactory`
  - `Session`
  - `Transaction`
  - `Query`

2. Configuration Basics

Two main approaches:

XML-based (Classic)

- `hibernate.cfg.xml` → contains DB connection properties.
- `.hbm.xml` → mapping file for classes and tables.

Annotation-based (Modern, preferred)

Use JPA annotations (`@Entity`, `@Table`, `@Id`, `@Column`, etc.) directly in Java classes.

Example:

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

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

    private int age;
}
```

3. Practical Usage

Learn how Hibernate is used in a real project:

1. **Basic CRUD Operations**
   - Save an object (`session.save(obj)`).
   - Retrieve object (`session.get()` / `session.load()`).
   - Update object (`session.update()`).
   - Delete object (`session.delete()`).
2. **Queries**
   - HQL: `from Student where age > 20`
   - Native SQL: `createSQLQuery("SELECT * FROM student")`
   - Criteria API: Dynamic query building.
3. **Relationships**
   - One-to-One (`@OneToOne`)
   - One-to-Many (`@OneToMany`)
   - Many-to-Many (`@ManyToMany`)
4. **Caching**
   - First-level cache (default, per session).
   - Second-level cache (shared across sessions).

4. Hands-On Practice

- Set up a small Maven project with Hibernate + MySQL/Oracle.
- Implement a `Student` entity.
- Perform full CRUD with Hibernate.
- Add relationships (e.g., Student–Course).
- Enable logging (`show_sql=true`) to see queries.

5. Interview Preparation

Expect questions like:

1. What is Hibernate, and why use it instead of JDBC?
2. Difference between `get()` and `load()`?
3. Explain lazy loading vs eager loading.
4. What are Hibernate caching levels?
5. What is the difference between HQL and SQL?
6. How does Hibernate manage transactions?

7. Difference between Session and SessionFactory?
8. How do you configure one-to-many mappings?
9. What is the N+1 select problem in Hibernate?
10. What strategies exist for primary key generation?

Recommended Path for You

1. Learn Hibernate basics (2–3 days).
2. Set up a mini project (CRUD + relationships).
3. Solve interview-style questions (theory + code snippets).

Hibernate project for a **Student–Course CRUD app**.

Files Included:

1. `hibernate.cfg.xml` (configuration file)
2. Entities (`Student.java`, `Course.java`) with **Many-to-Many relationship**
3. A runner (`MainApp.java`) to perform CRUD operations

Project Structure

```
hibernate-mini-project/
 ├── src/main/java/
 │    └── com/example/hibernate/
 │         ├── Student.java
 │         ├── Course.java
 │         └── MainApp.java
 └── src/main/resources/
      └── hibernate.cfg.xml
```

1. `hibernate.cfg.xml`

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!-- DB connection settings -->
        <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Drive
r</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibe
rnatedb?useSSL=false</property>
        <property
name="hibernate.connection.username">root</property>
        <property
name="hibernate.connection.password">password</property>

        <!-- Hibernate settings -->
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</pr
operty>
        <property name="hbm2ddl.auto">update</property>
        <property name="show_sql">true</property>
```

```
        <!-- Mapped classes -->
        <mapping class="com.example.hibernate.Student"/>
        <mapping class="com.example.hibernate.Course"/>
    </session-factory>
</hibernate-configuration>
```

2. Student.java

```
package com.example.hibernate;

import jakarta.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "student")
public class Student {

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

    private String name;

    private int age;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(
        name = "student_course",
        joinColumns = @JoinColumn(name = "student_id"),
        inverseJoinColumns = @JoinColumn(name = "course_id")
    )
    private Set<Course> courses = new HashSet<>();

    // Constructors
    public Student() {}
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getters and Setters
    public int getId() { return id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
```

```
    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
    public Set<Course> getCourses() { return courses; }
    public void setCourses(Set<Course> courses) { this.courses =
courses; }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + ", age="
+ age + "]";
    }
}
```

3. Course.java

```
package com.example.hibernate;

import jakarta.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "course")
public class Course {

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

    private String title;

    private int credits;

    @ManyToMany(mappedBy = "courses")
    private Set<Student> students = new HashSet<>();

    // Constructors
    public Course() {}
    public Course(String title, int credits) {
        this.title = title;
        this.credits = credits;
    }

    // Getters and Setters
    public int getId() { return id; }
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
```

```
    public int getCredits() { return credits; }
    public void setCredits(int credits) { this.credits =
credits; }
    public Set<Student> getStudents() { return students; }

    @Override
    public String toString() {
        return "Course [id=" + id + ", title=" + title + ",
credits=" + credits + "]";
    }
}
```

4. MainApp.java

```
package com.example.hibernate;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class MainApp {
    public static void main(String[] args) {

        // Load configuration
        Configuration cfg = new
Configuration().configure("hibernate.cfg.xml");
        SessionFactory factory = cfg.buildSessionFactory();
        Session session = factory.openSession();

        Transaction tx = session.beginTransaction();

        // Create Students
        Student s1 = new Student("Alice", 22);
        Student s2 = new Student("Bob", 24);

        // Create Courses
        Course c1 = new Course("Hibernate Basics", 3);
        Course c2 = new Course("Spring Boot", 4);

        // Establish relationships
        s1.getCourses().add(c1);
        s1.getCourses().add(c2);
        s2.getCourses().add(c1);

        // Save objects
        session.persist(s1);
        session.persist(s2);
```

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

        System.out.println("Data saved successfully!");
    }
}
```

What Happens Here?

- Tables `student`, `course`, and join table `student_course` will be created automatically.
- Running `MainApp` inserts sample students and courses.
- Relationships (many-to-many) are handled by Hibernate automatically.

Next Steps for you:

1. Run the project → see Hibernate logs + tables created.
2. Add CRUD operations (update a Student, delete a Course).
3. Try HQL queries like `from Student where age > 22`.

Hibernate Student–Course project with reusable **CRUD helper methods** (e.g., `addStudent`, `updateStudent`, `deleteStudent`, `listStudents`)

We'll create a **DAO-style utility class** (`HibernateUtil`) for CRUD operations on `Student`. You can replicate the same pattern for `Course`.

1. `HibernateUtil.java` (Helper Class for CRUD)

```
package com.example.hibernate;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import java.util.List;

public class HibernateUtil {

    private static final SessionFactory factory;

    static {
        try {
            factory = new Configuration()
                    .configure("hibernate.cfg.xml")
                    .addAnnotatedClass(Student.class)
                    .addAnnotatedClass(Course.class)
                    .buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    // Save Student
    public static void addStudent(Student student) {
        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();
        session.persist(student);
        tx.commit();
        session.close();
        System.out.println("Student added: " + student);
    }

    // Read Student by ID
    public static Student getStudent(int id) {
        Session session = factory.openSession();
        Student student = session.get(Student.class, id);
```

```
        session.close();
        return student;
    }

    // Update Student
    public static void updateStudent(Student student) {
        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();
        session.merge(student);
        tx.commit();
        session.close();
        System.out.println("Student updated: " + student);
    }

    // Delete Student
    public static void deleteStudent(int id) {
        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();
        Student student = session.get(Student.class, id);
        if (student != null) {
            session.remove(student);
            System.out.println("  Student deleted: " + student);
        }
        tx.commit();
        session.close();
    }

    // List all Students
    @SuppressWarnings("unchecked")
    public static List<Student> listStudents() {
        Session session = factory.openSession();
        List<Student> students = session.createQuery("from
Student").list();
        session.close();
        return students;
    }

    // Close Factory
    public static void shutdown() {
        factory.close();
    }
}
```

2. Updated `MainApp.java`

Now, instead of writing CRUD inside `main()`, we use helper methods:

```
package com.example.hibernate;

import java.util.List;

public class MainApp {
    public static void main(String[] args) {

        // Create Students
        Student s1 = new Student("Alice", 22);
        Student s2 = new Student("Bob", 24);

        // Create Courses
        Course c1 = new Course("Hibernate Basics", 3);
        Course c2 = new Course("Spring Boot", 4);

        // Establish relationships
        s1.getCourses().add(c1);
        s1.getCourses().add(c2);
        s2.getCourses().add(c1);

        // Save Students
        HibernateUtil.addStudent(s1);
        HibernateUtil.addStudent(s2);

        // List Students
        List<Student> students = HibernateUtil.listStudents();
        System.out.println("\n All Students:");
        students.forEach(System.out::println);

        // Get Student by ID
        Student fetched = HibernateUtil.getStudent(1);
        System.out.println("\n Fetched Student: " + fetched);

        // Update Student
        if (fetched != null) {
            fetched.setAge(25);
            HibernateUtil.updateStudent(fetched);
        }

        // Delete Student
        HibernateUtil.deleteStudent(2);

        // Final List
        students = HibernateUtil.listStudents();
        System.out.println("\n Students after delete:");
        students.forEach(System.out::println);
```

```
        HibernateUtil.shutdown();
    }
}
```

What You'll See When Running

1. Tables (`student`, `course`, `student_course`) created automatically.
2. Students inserted (`Alice`, `Bob`).
3. Fetch → update → delete → list operations executed with Hibernate logs printed.
4. Clear console output like:

```
Student added: Student [id=1, name=Alice, age=22]
Student added: Student [id=2, name=Bob, age=24]

All Students:
Student [id=1, name=Alice, age=22]
Student [id=2, name=Bob, age=24]

Fetched Student: Student [id=1, name=Alice, age=22]
Student updated: Student [id=1, name=Alice, age=25]
Student deleted: Student [id=2, name=Bob, age=24]

Students after delete:
Student [id=1, name=Alice, age=25]
```

With this setup, you have a **mini Hibernate CRUD app** that demonstrates:

- Entity mapping (`@ManyToMany`)
- Automatic table creation
- Full CRUD operations (Add, Get, Update, Delete, List)

CRUD methods for `Course` (so as to directly add/update/delete courses independent of students).

Updated `HibernateUtil.java` with Course CRUD

```
package com.example.hibernate;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import java.util.List;

public class HibernateUtil {

    private static final SessionFactory factory;

    static {
        try {
            factory = new Configuration()
                    .configure("hibernate.cfg.xml")
                    .addAnnotatedClass(Student.class)
                    .addAnnotatedClass(Course.class)
                    .buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    /* -------------------- STUDENT CRUD -------------------- */

    public static void addStudent(Student student) {
        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();
        session.persist(student);
        tx.commit();
        session.close();
        System.out.println(" Student added: " + student);
    }

    public static Student getStudent(int id) {
        Session session = factory.openSession();
        Student student = session.get(Student.class, id);
        session.close();
        return student;
    }
```

```
    public static void updateStudent(Student student) {
        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();
        session.merge(student);
        tx.commit();
        session.close();
        System.out.println(" Student updated: " + student);
    }

    public static void deleteStudent(int id) {
        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();
        Student student = session.get(Student.class, id);
        if (student != null) {
            session.remove(student);
            System.out.println(" Student deleted: " + student);
        }
        tx.commit();
        session.close();
    }

    @SuppressWarnings("unchecked")
    public static List<Student> listStudents() {
        Session session = factory.openSession();
        List<Student> students = session.createQuery("from
Student").list();
        session.close();
        return students;
    }

    /* -------------------- COURSE CRUD -------------------- */

    public static void addCourse(Course course) {
        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();
        session.persist(course);
        tx.commit();
        session.close();
        System.out.println(" Course added: " + course);
    }

    public static Course getCourse(int id) {
        Session session = factory.openSession();
        Course course = session.get(Course.class, id);
        session.close();
        return course;
```

```
    }

    public static void updateCourse(Course course) {
        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();
        session.merge(course);
        tx.commit();
        session.close();
        System.out.println(" Course updated: " + course);
    }

    public static void deleteCourse(int id) {
        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();
        Course course = session.get(Course.class, id);
        if (course != null) {
            session.remove(course);
            System.out.println(" Course deleted: " + course);
        }
        tx.commit();
        session.close();
    }

    @SuppressWarnings("unchecked")
    public static List<Course> listCourses() {
        Session session = factory.openSession();
        List<Course> courses = session.createQuery("from
Course").list();
        session.close();
        return courses;
    }

    /* -------------------- SHUTDOWN -------------------- */
    public static void shutdown() {
        factory.close();
    }
}
```

Example Usage in `MainApp.java`

```
package com.example.hibernate;

import java.util.List;

public class MainApp {
    public static void main(String[] args) {
```

```
        // --- Work with Students ---
        Student s1 = new Student("Alice", 22);
        Student s2 = new Student("Bob", 24);

        HibernateUtil.addStudent(s1);
        HibernateUtil.addStudent(s2);

        List<Student> students = HibernateUtil.listStudents();
        System.out.println("\n Students:");
        students.forEach(System.out::println);

        // --- Work with Courses ---
        Course c1 = new Course("Hibernate Basics", 3);
        Course c2 = new Course("Spring Boot", 4);

        HibernateUtil.addCourse(c1);
        HibernateUtil.addCourse(c2);

        List<Course> courses = HibernateUtil.listCourses();
        System.out.println("\n Courses:");
        courses.forEach(System.out::println);

        // Fetch Course
        Course fetchedCourse = HibernateUtil.getCourse(1);
        System.out.println("\n Fetched Course: " +
fetchedCourse);

        // Update Course
        if (fetchedCourse != null) {
            fetchedCourse.setCredits(5);
            HibernateUtil.updateCourse(fetchedCourse);
        }

        // Delete Course
        HibernateUtil.deleteCourse(2);

        courses = HibernateUtil.listCourses();
        System.out.println("\n Courses after delete:");
        courses.forEach(System.out::println);

        HibernateUtil.shutdown();
    }
}
```

With this extension, you now have **independent CRUD operations for both Students and Courses**.

- You can manage them separately.
- The relationship (`@ManyToMany`) will still link them automatically when you assign students to courses.

Further you can also **add a method that enrolls a Student into a Course (managing the `@ManyToMany` join table).**

C**omplete Student–Course Management System** with:

- **Entities:** `Student`, `Course` with `@ManyToMany` mapping.
- **HibernateUtil helper:** CRUD for both + **enrollStudentInCourse** method.
- **Main runner:** Shows how to create, fetch, update, delete, and enroll students into courses.

Full Hibernate Project Example

1. `Student.java`

```
package com.example.hibernate;

import jakarta.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "students")
public class Student {

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

    private String name;
    private int age;

    @ManyToMany(cascade = {CascadeType.ALL}, fetch =
FetchType.LAZY)
    @JoinTable(
            name = "student_course",
            joinColumns = {@JoinColumn(name = "student_id")},
            inverseJoinColumns = {@JoinColumn(name =
"course_id")}
    )
    private Set<Course> courses = new HashSet<>();

    public Student() {}
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
```

```
    // Getters and setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }

    public Set<Course> getCourses() { return courses; }
    public void setCourses(Set<Course> courses) { this.courses =
courses; }

    @Override
    public String toString() {
        return "Student{id=" + id + ", name='" + name + "',
age=" + age + "}";
    }
}
```

2. Course.java

```
package com.example.hibernate;

import jakarta.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "courses")
public class Course {

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

    private String title;
    private int credits;

    @ManyToMany(mappedBy = "courses", fetch = FetchType.LAZY)
    private Set<Student> students = new HashSet<>();

    public Course() {}
    public Course(String title, int credits) {
```

```
        this.title = title;
        this.credits = credits;
    }

    // Getters and setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public int getCredits() { return credits; }
    public void setCredits(int credits) { this.credits =
credits; }

    public Set<Student> getStudents() { return students; }
    public void setStudents(Set<Student> students) {
this.students = students; }

    @Override
    public String toString() {
        return "Course{id=" + id + ", title='" + title + "',
credits=" + credits + "}";
    }
}
```

3. HibernateUtil.java

```
package com.example.hibernate;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import java.util.List;

public class HibernateUtil {

    private static final SessionFactory factory;

    static {
        try {
            factory = new Configuration()
                    .configure("hibernate.cfg.xml")
                    .addAnnotatedClass(Student.class)
```

```
                    .addAnnotatedClass(Course.class)
                    .buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    /* ---------------- STUDENT CRUD ---------------- */

    public static void addStudent(Student student) {
        try (Session session = factory.openSession()) {
            Transaction tx = session.beginTransaction();
            session.persist(student);
            tx.commit();
        }
    }

    public static Student getStudent(int id) {
        try (Session session = factory.openSession()) {
            return session.get(Student.class, id);
        }
    }

    public static void updateStudent(Student student) {
        try (Session session = factory.openSession()) {
            Transaction tx = session.beginTransaction();
            session.merge(student);
            tx.commit();
        }
    }

    public static void deleteStudent(int id) {
        try (Session session = factory.openSession()) {
            Transaction tx = session.beginTransaction();
            Student student = session.get(Student.class, id);
            if (student != null) session.remove(student);
            tx.commit();
        }
    }

    public static List<Student> listStudents() {
        try (Session session = factory.openSession()) {
            return session.createQuery("from Student",
Student.class).list();
        }
    }
```

```
    /* ---------------- COURSE CRUD ---------------- */

    public static void addCourse(Course course) {
        try (Session session = factory.openSession()) {
            Transaction tx = session.beginTransaction();
            session.persist(course);
            tx.commit();
        }
    }

    public static Course getCourse(int id) {
        try (Session session = factory.openSession()) {
            return session.get(Course.class, id);
        }
    }

    public static void updateCourse(Course course) {
        try (Session session = factory.openSession()) {
            Transaction tx = session.beginTransaction();
            session.merge(course);
            tx.commit();
        }
    }

    public static void deleteCourse(int id) {
        try (Session session = factory.openSession()) {
            Transaction tx = session.beginTransaction();
            Course course = session.get(Course.class, id);
            if (course != null) session.remove(course);
            tx.commit();
        }
    }

    public static List<Course> listCourses() {
        try (Session session = factory.openSession()) {
            return session.createQuery("from Course",
Course.class).list();
        }
    }

    /* ---------------- ENROLL STUDENT IN COURSE ---------------
- */

    public static void enrollStudentInCourse(int studentId, int
courseId) {
        try (Session session = factory.openSession()) {
            Transaction tx = session.beginTransaction();
```

```
            Student student = session.get(Student.class,
studentId);
            Course course = session.get(Course.class, courseId);

            if (student != null && course != null) {
                student.getCourses().add(course);
                course.getStudents().add(student);

                session.merge(student); // update the join table
                System.out.println(" Enrolled " +
student.getName() + " into " + course.getTitle());
            } else {
                System.out.println("□□ Student or Course not
found!");
            }

            tx.commit();
        }
    }

    /* ---------------- SHUTDOWN ---------------- */
    public static void shutdown() {
        factory.close();
    }
}
```

4. MainApp.java

```
package com.example.hibernate;

public class MainApp {
    public static void main(String[] args) {

        // --- Add Students ---
        Student s1 = new Student("Alice", 22);
        Student s2 = new Student("Bob", 24);
        HibernateUtil.addStudent(s1);
        HibernateUtil.addStudent(s2);

        // --- Add Courses ---
        Course c1 = new Course("Hibernate Basics", 3);
        Course c2 = new Course("Spring Boot", 4);
        HibernateUtil.addCourse(c1);
        HibernateUtil.addCourse(c2);
```

```
        // --- Enroll Students into Courses ---
        HibernateUtil.enrollStudentInCourse(s1.getId(),
c1.getId());
        HibernateUtil.enrollStudentInCourse(s1.getId(),
c2.getId());
        HibernateUtil.enrollStudentInCourse(s2.getId(),
c1.getId());

        // --- List Students ---
        System.out.println("\n Students:");
        HibernateUtil.listStudents().forEach(student ->
                System.out.println(student + " -> Courses: " +
student.getCourses())
        );

        // --- List Courses ---
        System.out.println("\n Courses:");
        HibernateUtil.listCourses().forEach(course ->
                System.out.println(course + " -> Students: " +
course.getStudents())
        );

        HibernateUtil.shutdown();
    }
}
```

With this, you now have:

- **Independent CRUD** for Students & Courses
- **Many-to-Many Join Table** managed automatically
- **Method to enroll Students in Courses** (`student_course` join table populated)
- A **working runner** that demonstrates everything.