# Hibernate Lab

**Theory :**

**JDBC** : JDBC stands for Java Database Connectivity. It is a java application programming interface to provide a connection between the Java programming language and a wide range of databases, it establishes a link between them, so that a programmer could send data from Java code and store it in the database for future use.

**Hibernate** : Hibernate is a high-performance open source Object-Relational Mapping framework and query service. Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities. **Object-Relational Mapping (ORM)** is the process of converting Java objects to database tables. In other words, this allows us to interact with a relational database without any SQL. Simply, It is mapping between object and relational tuples

**JDBC v/s Hibernate**
Hibernate performs an object-relational mapping framework, while JDBC is simply a database connectivity API.

Why Hibernate?

- Hibernate supports the mapping of java classes to database tables and vice versa. It provides features to perform CRUD operations across all the major relational databases.
- Hibernate eliminates all the boiler-plate code that comes with JDBC and takes care of managing resources, so we can focus on business use cases rather than making sure that database operations are not causing resource leaks.
- Hibernate supports transaction management and makes sure there is no inconsistent data present in the system.

The Java Persistence API (JPA) is a specification that defines how to persist data in Java applications. Hibernate is an implementation of JPA guidelines. It helps in mapping Java data objects types to SQL data types. The primary focus of JPA is the ORM layer.

| JPA | Hibernate |
|---|---|
| Java Persistence API - It is not an implementation. It is only a Java specification. | Hibernate is an implementation of JPA. Hence, the common standard which is given by JPA is followed by Hibernate. |

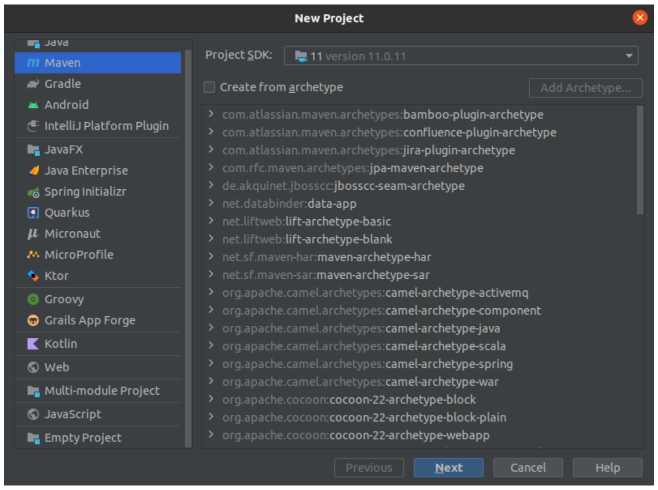
**Lab Activities :** (Hands-On)
Prerequisite -

- MySQL
- IntelliJ IDE
- Java 8 / 11

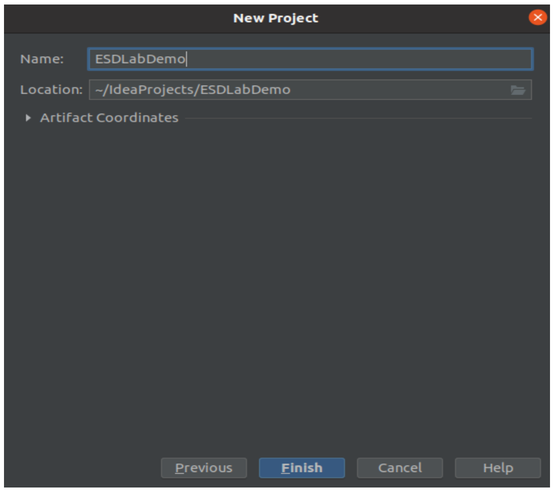
GitHub Repo for Hands-On session - https://github.com/NirajGujarathi/ESDLabDemo

**Steps →**

1. Create Maven Project
    - File
    - New Project
    - Select Maven Project
    - Select Project SDK (java 1.8(java 8) / java 11)
    - Assign Project / Artifact name
    - Click Finish


New Project
Java
Maven
Gradle
Android
IntelliJ Platform Plugin
JavaFX
Java Enterprise
Spring Initializr
Quarkus
Micronaut
MicroProfile
Ktor
Groovy
Grails App Forge
Kotlin
Web
Multi-module Project
JavaScript
Empty Project
Create from archetype
Add Archetype...
com.atlassian.maven.archetypes:bamboo-plugin-archetype
com.atlassian.maven.archetypes:confluence-plugin-archetype
com.atlassian.maven.archetypes:jira-plugin-archetype
com.rfc.maven.archetypes:jpa-maven-archetype
de.akquinet.jbosscc:jbosscc-seam-archetype
net.databinder:data-app
net.liftweb:lift-archetype-basic
net.liftweb:lift-archetype-blank
net.sf.maven-har:maven-archetype-har
net.sf.maven-sar:maven-archetype-sar
org.apache.camel.archetypes:camel-archetype-activemq
org.apache.camel.archetypes:camel-archetype-component
org.apache.camel.archetypes:camel-archetype-java
org.apache.camel.archetypes:camel-archetype-scala
org.apache.camel.archetypes:camel-archetype-spring
org.apache.camel.archetypes:camel-archetype-war
org.apache.cocoon:cocoon-22-archetype-block
org.apache.cocoon:cocoon-22-archetype-block-plain
org.apache.cocoon:cocoon-22-archetype-webapp
Previous
Next
Cancel
Help

Project SDK: 11 version 11.0.11



Name: ESDLabDemo
Location: ~/IdeaProjects/ESDLabDemo

2. Add hibernate-core and mysql-connector dependency in pom.xml file

    Following 2 dependencies are required
    **hibernate-core** contains all the core hibernate classes, so we will get all the necessary features of hibernate and hibernate annotations in your project.
    https://mvnrepository.com/artifact/org.hibernate/hibernate-core/6.1.4.Final

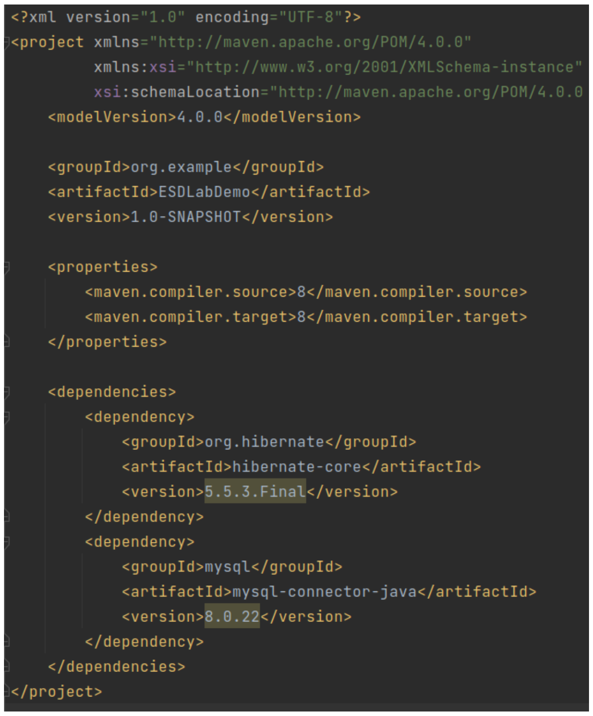
    **mysql-connector-java** is the MySQL driver for connecting to MySQL databases, if you are using any other database then add corresponding DBMS driver artifacts.
    https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.31

Add these code snippets after </properties> tag for (**java 11+** versions) and use

```
<dependencies>
      <dependency>
          <groupId>org.hibernate</groupId>
          <artifactId>hibernate-core</artifactId>
          <version>6.1.4.Final</version>
      </dependency>
      <dependency>
          <groupId>mysql</groupId>
          <artifactId>mysql-connector-java</artifactId>
          <version>8.0.31</version>
      </dependency>
</dependencies>
```

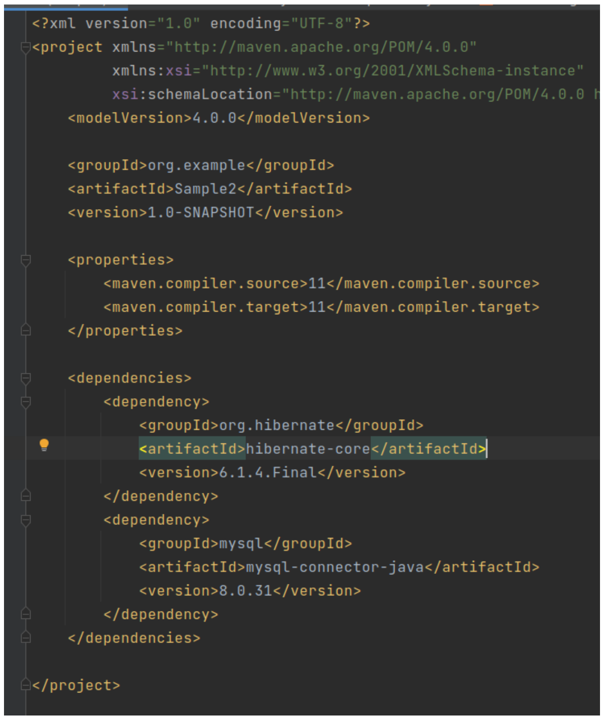
For java 8 change **hibernate** version to `<version> 5.5.3.Final </version>` and **mysql-connector** to `<version> 8.0.22 </version>`

Java 8



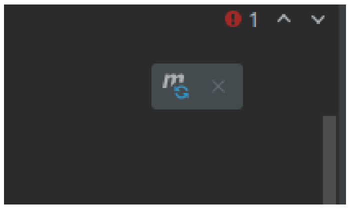
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>ESDLabDemo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.5.3.Final</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.22</version>
        </dependency>
    </dependencies>
</project>
```

Java 11



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 h
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>Sample2</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>6.1.4.Final</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.31</version>
        </dependency>
    </dependencies>

</project>
```

After adding these dependency you have to sync it with local maven repository - it can be download and synced automatically; else you can sync these dependency manually with button on top right corner in pom.xml



All maven dependencies should be reflected in external libraries

## 3. Create Bean class which is entity class in hibernate

- Create class Department
- Annotate the class as @**Entity** and specify @**Table** name
- import hibernate annotations → press (alt+Enter) for auto import
  - **javax.persistence.* (java 8 and hibernate 5.5)**
  - **jakarta.persistence.*  (java 11 and hibernate 6.1)**
- Add respective class attributes these are column names in your MySQL relational table
- departmentID - is your primary key for relational table → annotated with @**Id** and @**GeneratedValue** defines sequencing strategy for ID generation
- departmentName and deptAddress are @**Column** in SQL table

```
package com.example.esd.Bean;
import jakarta.persistence.*;
@Entity
@Table(name ="department")
public class Department {
  @Id
  @Column(name ="dept_id")
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int departmentID;

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

  @Column(name="dept_address")
  private String deptAddress;
}
```

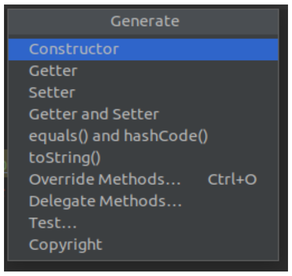
Hibernate Entity class Annotations →

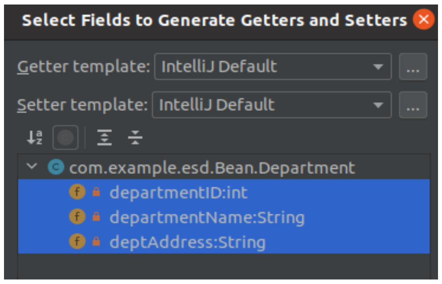
- @**Entity** annotation marks this class as an entity.
- @**Table** annotation specifies the table name where data of this entity is to be persisted. If you don't use @Table annotation, hibernate will use the class name as the table name by default.
- @**Id** annotation marks the identifier for this entity.
- @**Column** annotation specifies the details of the column for this property or field. If @Column annotation is not specified, property name will be used as the column name by default.

- @**GeneratedValue** annotation is useful to define the strategy to create unique primary key values. You can choose between SEQUENCE, IDENTITY, TABLE, and AUTO strategy. This annotation helps in creating surrogate keys for a table, it is a column with a unique identifier for each row. https://www.tutorialandexample.com/hibernate-generatedvalue-strategies/

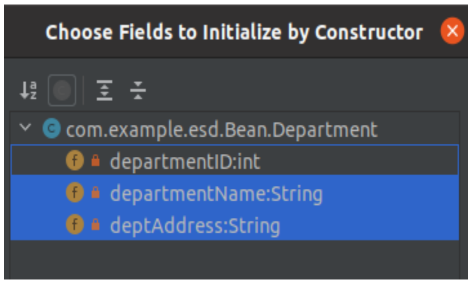
- As well as you can add validation constraints to Entity, such as **@NotNull**, **@Check()** → https://hibernate.org/validator/documentation/getting-started/

- Now, add constructor and getter/ setter for these attributes → right click on IDE screen generate (alt+Insert) you will get screen to auto generate parameterized constructors and getters/ setters





Getter template: IntelliJ Default
Setter template: IntelliJ Default
com.example.esd.Bean.Department
departmentID:int
departmentName:String
deptAddress:String



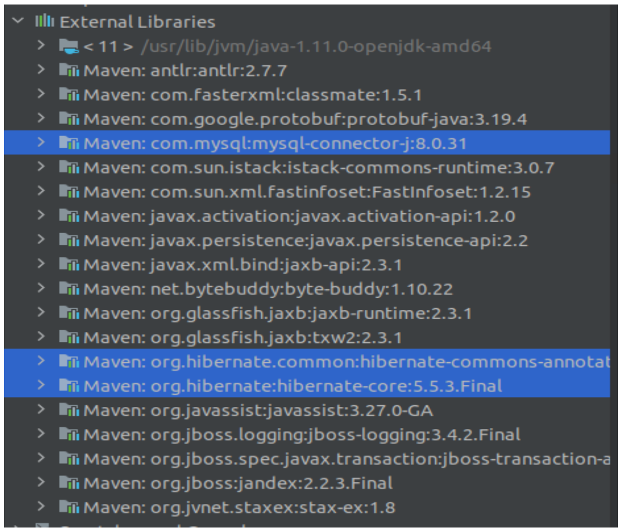
com.example.esd.Bean.Department
departmentID:int
departmentName:String
deptAddress:String

## 4. Create Hibernate Configuration file

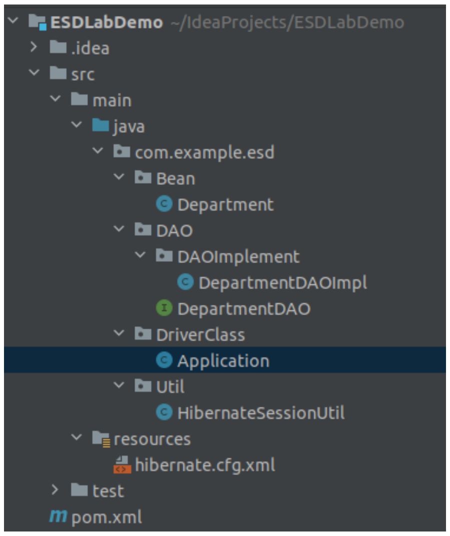
- create **hibernate.cfg.xml** under **src/main/resources**,
- Right Click on Resources folder and create new file - name it as **hibernate.cfg.xml**
- It is used to add database credentials and mapping relational tables with java classes.
- Add following code snippet and **change username and password according to your mysql credentials**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
      <property
name="connection.url">jdbc:mysql://localhost:3306/DemoDB?createDatabaseIfNotExi
st=true</property>
      <property name="connection.username">niraj</property>
      <property name="connection.password">password</property>
      <property
name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>

      <!-- DB schema will be updated if needed -->
      <property name="hibernate.hbm2ddl.auto">update</property>
      <property name="show_sql">true</property>
      <mapping class="com.example.esd.Bean.Department"/>
  </session-factory>
</hibernate-configuration>
```

If you are getting any error in **hibernate.cfg.xml** file,
In the property name="connection.driver_class" → **reload the project, else close and reopen.**
Intellij might take time in loading hibernate and mysql connector related libraries
At the end these 3 external maven libraries should be there in your project.


External Libraries
< 11 > /usr/lib/jvm/java-1.11.0-openjdk-amd64
Maven: antlr:antlr:2.7.7
Maven: com.fasterxml:classmate:1.5.1
Maven: com.google.protobuf:protobuf-java:3.19.4
Maven: com.mysql:mysql-connector-j:8.0.31
Maven: com.sun.istack:istack-commons-runtime:3.0.7
Maven: com.sun.xml.fastinfoset:FastInfoset:1.2.15
Maven: javax.activation:javax.activation-api:1.2.0
Maven: javax.persistence:javax.persistence-api:2.2
Maven: javax.xml.bind:jaxb-api:2.3.1
Maven: net.bytebuddy:byte-buddy:1.10.22
Maven: org.glassfish.jaxb:jaxb-runtime:2.3.1
Maven: org.glassfish.jaxb:txw2:2.3.1
Maven: org.hibernate.common:hibernate-commons-annotat
Maven: org.hibernate:hibernate-core:5.5.3.Final
Maven: org.javassist:javassist:3.27.0-GA
Maven: org.jboss.logging:jboss-logging:3.4.2.Final
Maven: org.jboss.spec.javax.transaction:jboss-transaction-a
Maven: org.jboss:jandex:2.2.3.Final
Maven: org.jvnet.staxex:stax-ex:1.8

5. Maintain project structure (Packaging each layer)


```
ESDLabDemo ~/IdeaProjects/ESDLabDemo
  .idea
  src
    main
      java
        com.example.esd
          Bean
            Department
          DAO
            DAOImplement
              DepartmentDAOImpl
            DepartmentDAO
          DriverClass
            Application
          Util
            HibernateSessionUtil
      resources
        hibernate.cfg.xml
    test
  pom.xml
```

6. Create Hibernate SessionFactory

Create class file under **src/main/java/com/example/esd/Util** folder - this class provides single instance of session object for database - it follows singleton design pattern

```
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
```

```java
public class HibernateSessionUtil {

  //XML based configuration
  private static final SessionFactory sessionFactory;

  static {
      try {
          Configuration configuration = new Configuration();
          configuration.configure();

          sessionFactory = configuration.buildSessionFactory();
      } catch (Throwable ex) {
          throw new ExceptionInInitializerError(ex);
      }
  }

  public static Session getSession() throws HibernateException {
      return sessionFactory.openSession();
  }
}
```

7. Create DAO folder and DAOImplementation

a) Department DAO Interface -

```java
package com.example.esd.DAO;
import com.example.esd.Bean.Department;
public interface DepartmentDAO {
  boolean addDepartment(Department deptObj);
}
```

b) Department DAO Implementation -

```java
public class DepartmentDAOImpl implements DepartmentDAO {
   @Override
   public boolean addDepartment(Department deptObj) {
       try(Session session = HibernateSessionUtil.getSession()){
           Transaction transaction = session.beginTransaction();
           session.persist(deptObj);
           transaction.commit();
           return true;
       }
       catch (HibernateException exception) {
           System.out.println("Hibernate Exception");
           System.out.print(exception.getLocalizedMessage());
           return false;
       }
   }
}
```

- DAO class is main crux of Hibernate where Java Object are mapped with MySQL tables
- Hibernate is tool for JPA - java Persistence API - with the help of this hibernate framework can persist / store java object by mapping them with relational tables

## 8. Create Application class & Run the Application

```java
public class Application {
   public static void main(String args[]){
       System.out.println("Application Started");
       runApplication();
       System.out.println("End");
   }
   private static void runApplication() {
       Department dept1= new Department("Development","Mumbai");
       Department dept2= new Department();
       dept2.setDepartmentName("DataScience");
       dept2.setDeptAddress("Bangalore");
       DepartmentDAO deptDAO= new DepartmentDAOImpl();
       if(deptDAO.addDepartment(dept1)){
           System.out.println("department 1 added Successfully");
       }
       if(deptDAO.addDepartment(dept2)){
           System.out.println("department 2 added Successfully");
       }
   }
}
```

It is a simple use case of Hibernate framework where we have mapped departement java objects to MySQL relations and saved these objects into databases.

In the following GitHub repository I have created a project considering a companydb use case. It includes hibernate association mapping among employees and departments (ManyToOne/ OneToMany) ; along with that it includes all **add, retrieve, delete** operations on **employee, department and project entities**, using hibernate and HQL (Hibernate Query language).

**GitHub Link – https://github.com/NirajGujarathi/HibernateLab**

## Hibernate Mapping

#12 Hibernate Tutorial | Mapping Relations Theory

- For multiple Entities we have to map these entities by @OneToMany or @ManyToOne mappings

- There is **@ManyToOne** mapping between employees and departments, many employees can have one department.

```
@ManyToOne
@JoinColumn(name="employee_dept_id")
private Department department;
```

- Similarly, One department can have multiple employees we will use **@OneToMany**

```
@OneToMany(mappedBy = "department", fetch = FetchType.EAGER)
// name of class member variable in Employee class; it will be mapped with
that variable
@JsonIgnore
private List<Employee> employeesList;
```

- These are mapping annotations useful in implementing relations amongst entities. https://www.javatpoint.com/hibernate-many-to-many-example-using-annotation

Youtube Reference Video for Hibernate Mapping

#13 Hibernate Tutorial | Mapping Relations Practical
#14 Hibernate Tutorial | Fetch EAGER LAZY

In case java objects fail to lazily initialize a collection of roles; add EAGER fetching `fetch = FetchType.EAGER` for @OneToMany mapping. By default child objects are fetched lazily.

- **Eager Loading** is a design pattern in which data initialization occurs on the spot
- **Lazy Loading** is a design pattern which is used to defer initialization of an object as long as it's possible

## Reference Links :

**Complete Project GitHub Link** – https://github.com/NirajGujarathi/HibernateLab

https://thorben-janssen.com/5-things-you-need-to-know-when-using-hibernate-with-mysql/
https://www.geeksforgeeks.org/hibernate-annotations/
https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/