Hibernate

**Hibernate Framework**

**Hibernate is a Java framework that simplifies the development of Java application to interact with the database**. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

**Data Persistence** :- Representing data permanently in backend systems is called “Data Persistence”.

**ORM Tool**

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.



The ORM tool internally uses the JDBC API to interact with the database.

**What is JPA?**

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

**Advantages of Hibernate Framework**

Following are the advantages of hibernate framework:

**1) Open Source and Lightweight**

Hibernate framework is open source and lightweight.

**2) Fast Performance**

The performance of hibernate framework is fast because cache is internally used in hibernate framework.

**3) Database Independent Query**

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

**4) Automatic Table Creation**

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

**5) Simplifies Complex Join**

Fetching data from multiple tables is easy in hibernate framework.

**Hibernate Architecture**

The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

The Hibernate architecture is categorized in four layers.

* Java application layer
* Hibernate framework layer
* Backend API layer
* Database layer

Let's see the diagram of hibernate architecture:



This is the high level architecture of Hibernate with mapping file and configuration file.



Hibernate framework uses many objects such as session factory, session, transaction etc. along with existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

**Elements of Hibernate Architecture**

|  |
| --- |
| For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows: |

#### **SessionFactory**

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

#### **Session**

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

#### **Transaction**

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

#### **ConnectionProvider**

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

#### **TransactionFactory**

It is a factory of Transaction. It is optional.

# Hibernate First Example

Here, we are going to create a simple example of hibernate application using eclipse IDE. For creating the first hibernate application in Eclipse IDE, we need to follow the following steps:

1. Create the java project
   1. Create the java project by **File Menu** - **New** - **project** - **java project**. Now specify the project name e.g. firsthb then **next**- **finish**.
2. Add jar files for hibernate
   1. To add the jar files **Right click on your project** - **Build path** - **Add external archives**. Now select all the jar files as shown in the image given below then click open.
3. Create the Persistent class
   1. Here, we are creating the same persistent class. To create the persistent class, Right click on **src** - **New** - **Class** - specify the class with package name (e.g. com.testpack.mypackage) – **finish**

|  |
| --- |
| **public** **class** Employee {  **private** **int** id;  **private** String firstName,lastName;    **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getFirstName() {  **return** firstName;  }  **public** **void** setFirstName(String firstName) {  **this**.firstName = firstName;  }  **public** String getLastName() {  **return** lastName;  }  **public** **void** setLastName(String lastName) {  **this**.lastName = lastName;  }  } |

1. Create the mapping file for Persistent class
   1. Here, we are creating the same mapping file. To create the mapping file, Right click on **src** - **new** - **file** - specify the file name (e.g. employee.hbm.xml) - **ok**. It must be outside the package.

#### **employee.hbm.xml**

|  |
| --- |
| <?xml version='1.0' encoding='UTF-8'?>  <!DOCTYPE hibernate-mapping PUBLIC   "-//Hibernate/Hibernate Mapping DTD 5.3//EN"   "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">     <hibernate-mapping>    <**class** name="com.mypackage.Employee" table="emp1000">      <id name="id">       <generator **class**="assigned"></generator>      </id>      <property name="firstName"></property>      <property name="lastName"></property>    </**class**>   </hibernate-mapping> |

1. Create the Configuration file
   1. The configuration file contains all the information’s for the database such as connection\_url, driver\_class, username, password etc. The **hbm2ddl.auto** property is used to create the table in the database automatically. We will have in-depth learning about Dialect class in next topics.
   2. To create the configuration file, right click on src - new - file. Now specify the configuration file name e.g. **hibernate.cfg.xml**

|  |
| --- |
| <?xml version='1.0' encoding='UTF-8'?>  <!DOCTYPE hibernate-configuration PUBLIC            "-//Hibernate/Hibernate Configuration DTD 5.3//EN"            "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">  <hibernate-configuration>      <session-factory>          <property name="hbm2ddl.auto">update</property>          <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>          <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>        <property name="connection.username">system</property>          <property name="connection.password">oracle</property>          <property name="connection.driver\_class">oracle.jdbc.driver.OracleDriver</property>      <mapping resource="employee.hbm.xml"/>      </session-factory>  </hibernate-configuration> |

For MySQL

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-configuration PUBLIC            "-//Hibernate/Hibernate Configuration DTD 5.3//EN"            "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">    <hibernate-configuration>  <session-factory>  <property name=*"connection.driver\_class"*>com.mysql.cj.jdbc.Driver</property>  <property name=*"connection.url"*>jdbc:mysql://localhost:3306/hyberdb</property>  <property name=*"dialect"*>org.hibernate.dialect.MySQL5Dialect</property>  <property name=*"connection.user"*>root</property>  <property name=*"connection.password"*>root</property>  <property name=*"show\_sql"*>true</property>  <property name=*"hbm2ddl.auto"*>update</property>  <mapping resource=*"emp.hbm.xml"*/>  </session-factory>  </hibernate-configuration> |

**hbm2ddl.auto**

**create** :- It create new table every time, before creating table, it will delete table if table already exist.

**create-drop** :- it create new table every time and after completion action it delete's that table automatically

**update** :- it creates new table if table not exist, if it exist just it will continue other operations.

1. Create the class that retrieves or stores the persistent object (testMain.java)

|  |
| --- |
| Configuration cfg = **new** Configuration();  cfg.configure("emp.cfg.xml");    SessionFactory sf = cfg.buildSessionFactory();    Session s = sf.openSession();  Transaction t = s.beginTransaction();  Employee emp = **new** Employee(1001, "Prateek");  Serializable sz = s.save(emp);  t.commit();  s.close();  sf.close();  System.***out***.println((Integer)sz + " Inserted....."); |

1. Run the application

Sample Main Class Code

|  |
| --- |
| Customer c1 = **new** Customer();  c1.setCid(1002);  c1.setCname("Pavan Kumar");  c1.setEmail("Pavan@gmail.com");  c1.setPhno("9988123256");  c1.setLoc("Hyderabad");  c1.setPincode("500090");  Date d1 =**new** Date();  c1.setcDate(d1);  Student s1 = **new** Student();  s1.setSno(122);  s1.setSname("Sahasra");  s1.setFees(12000.00f);    // persistant object    Configuration cfg = **new** Configuration();  cfg.configure("hibernate.cfg.xml");// reading content from config file  SessionFactory sf = cfg.buildSessionFactory(); // making connection to database  Session s = sf.openSession();  Transaction t = s.beginTransaction();  s.save(c1); // it will create table and stores data in table from c1 object.  s.save(s1);  t.commit();  sf.close();  s.close();    System.***out***.println("Rows are Inserted...."); |