**RDBMS**

**Ruchira Supipi Wilebadaarachchi.**

Contents

[1. Hibernate (javabrains) 4](#_Toc479329758)

[Introduction 4](#_Toc479329759)

[The problem 4](#_Toc479329760)

[Install Hibernate 4](#_Toc479329761)

[E-clips 4](#_Toc479329762)

[Netbeans – 5](#_Toc479329763)

[Using Hibernate 5](#_Toc479329764)

[Without hibernate 5](#_Toc479329765)

[With hibernate 5](#_Toc479329766)

[JDBC Database configuration 5](#_Toc479329767)

[The model object – annotations 5](#_Toc479329768)

[Using the hibernate API 7](#_Toc479329769)

[Hibernate – (TPoint) 9](#_Toc479329770)

[Advantages of Hibernate Framework 9](#_Toc479329771)

[Hibernate Architecture 9](#_Toc479329772)

[SessionFactory 11](#_Toc479329773)

[Transaction 11](#_Toc479329774)

IOC - Invers Of Control

AOP - Aspect Oriented Programming

ORM - Object relational Mapping

JSF - Java Server Faces

JSTL -

ORM - Object Relational Mapping

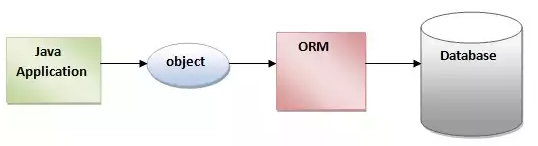
POJO - Plain Old java Object

POJI - Plain Old java Interface

ODBC - Open data base connectivity

# Hibernate – (TPoint)





Advantages of Hibernate Framework

There are many advantages of Hibernate Framework. They are as follows:

**1) Opensource and Lightweight:** Hibernate framework is opensource under the LGPL license and lightweight.

**2) Fast performance:** The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled bydefault.

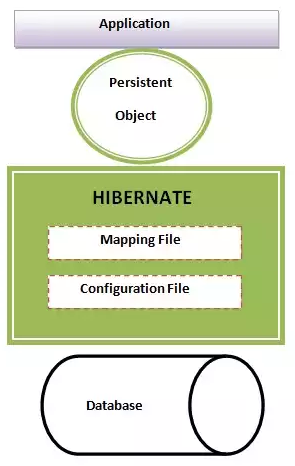
**3) Database Independent query:** HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, If database is changed for the project, we need to change the SQL query as well 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:** To fetch data form multiple tables is easy in hibernate framework.

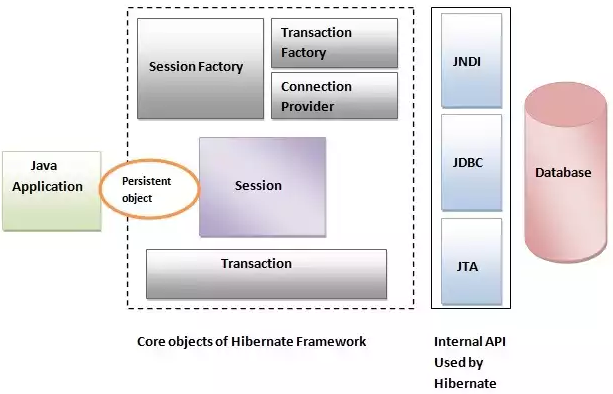
**6) Provides query statistics and database status:** Hibernate supports Query cache and provide statistics about query and database status.

## Hibernate Architecture



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

There are 4 layers in hibernate architecture java **application layer, hibernate framework layer, backhand api layer** and **database layer**.Let's see the diagram of hibernate architecture:



Hibernate framework uses many objects 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).

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.

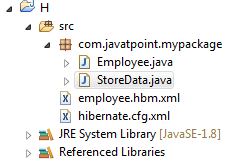
TransactionFactory

It is a factory of Transaction. It is optional.

## With XML file

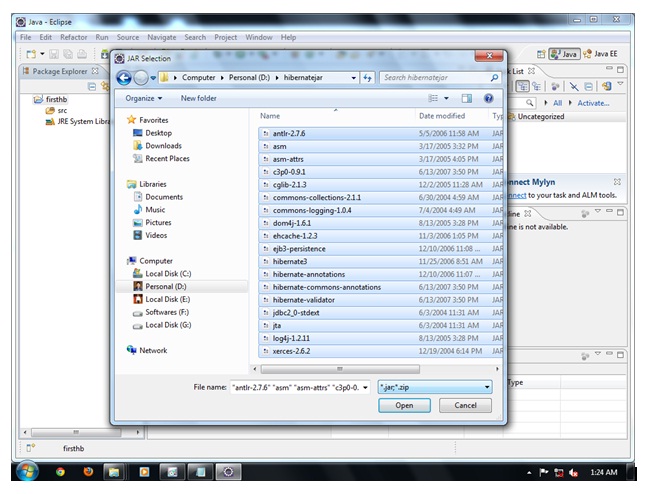
1. Create the java project

Create the java project by **File Menu** - **New** - **project** - **java project** . Now specify the project name e.g. firsthb then **next** - **finish** .



1. Add jar files for hibernate

**Right click on your project** - **Build path** - **Add external archives**



1. Create the Persistent class

**Employee.java**

**package** com.javatpoint.mypackage;

**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

**employee.hbm.xml**

<?xml version=*'1.0'* encoding=*'UTF-8'*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"com.javatpoint.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. Need to add mysql connector

**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">

<!-- Generated by MyEclipse Hibernate Tools. -->

<hibernate-configuration>

<session-factory>

<property name=*"hbm2ddl.auto"*>update</property>

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/office</property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.password"*>pipi</property>

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<mapping resource=*"employee.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

* If use oracle sql

<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>

1. Create the class that retrieves or stores the persistent object

**StoreData.java**

**package** com.javatpoint.mypackage;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.Configuration;

**public** **class** StoreData {

**public** **static** **void** main(String[] args) {

// creating configuration object

Configuration cfg = **new** Configuration();

cfg.configure("hibernate.cfg.xml");// populates the data of the

// configuration file

// creating seession factory object

SessionFactory factory = cfg.buildSessionFactory();

// creating session object

Session session = factory.openSession();

// creating transaction object

Transaction t = session.beginTransaction();

Employee e1 = **new** Employee();

e1.setId(101);

e1.setFirstName("ruchira1");

e1.setLastName("Supipi1");

session.persist(e1);// persisting the object

t.commit();// transaction is commited

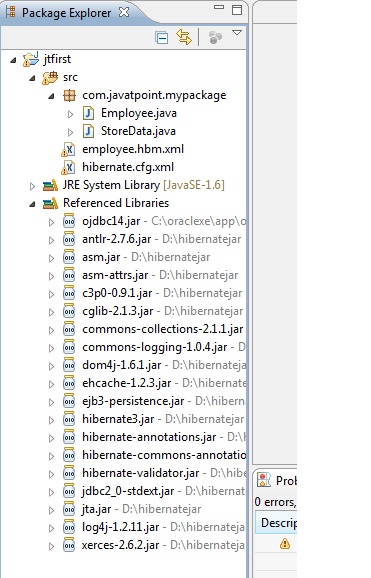
session.close();

System.***out***.println("successfully saved");

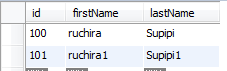
}

}

1. Run the application



Output:



## With Annotation

Hibernate Annotations are based on the JPA 2 specification and supports all the features.

All the JPA annotations are defined in the javax.persistence.\* package.

Hibernate **EntityManager** implements the interfaces and life cycle defined by the JPA specification.

The core advantage of using hibernate annotation is that you don't need to create mapping (hbm) file. Here, hibernate annotations are used to provide the meta data.

1. **Add the jar file for oracle (if your database is oracle) and annotation**

For oracle you need to add **ojdbc14.jar** file. For using annotation, you need to add:

* **hibernate-commons-annotations.jar**
* **ejb3-persistence.jar**
* **hibernate-annotations.jar**

1. **Create the Persistent class**

we are creating the same persistent class which we have created in the previous topic. But here, we are using annotation.

**@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 bydefault.

**@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 bydefault.

**Employee.java**

**package** com.javatpoint.mypackage;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name="emp001")

**public** **class** Employee {

@Id

**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. **Add mapping of Persistent class in configuration file**

<mapping class=*"com.javatpoint.mypackage.Employee"* />

open the hibernate.cgf.xml file, and add an entry of mapping resource

<?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">

<!-- Generated by MyEclipse Hibernate Tools. -->

<hibernate-configuration>

<session-factory>

<property name=*"hbm2ddl.auto"*>update</property>

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/office</property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.password"*>pipi</property>

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<mapping class=*"com.javatpoint.mypackage.Employee"* />

</session-factory>

</hibernate-configuration>

1. **Create the class that retrieves or stores the persistent object**

**StoreData.java**

**package** com.javatpoint.mypackage;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**public** **class** StoreData {

**public** **static** **void** main(String[] args) {

Session session = **new** AnnotationConfiguration().configure().buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

Employee e1 = **new** Employee();

e1.setId(1001);

e1.setFirstName("ruchi1");

e1.setLastName("Supipi1");

Employee e2 = **new** Employee();

e2.setId(1002);

e2.setFirstName("Ruchi2");

e2.setLastName("Supipi2");

session.persist(e1);

session.persist(e2);

t.commit();

session.close();

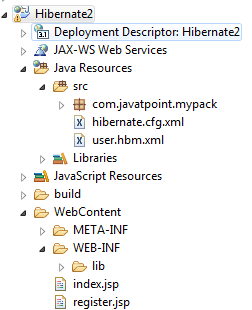
System.***out***.println("successfully saved");

}

}

## Web Application

1. Create a “Dynamic web application”



Add required .jar to the lib folder and add to the build path.

1. **index.jsp**

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<form action=*"register.jsp"* method=*"post"*>

Name:<input type=*"text"* name=*"name"* /><br>

<br /> Password:<input type=*"password"* name=*"password"* /><br>

<br /> Email ID:<input type=*"text"* name=*"email"* /><br>

<br /> <input type=*"submit"* value=*"register"* />

</form>

</body>

</html>

1. **register.jsp**

<%@page import=*"com.javatpoint.mypack.UserDao"*%>

<jsp:useBean id=*"obj"* class=*"com.javatpoint.mypack.User"*>

</jsp:useBean>

<jsp:setProperty property=*"\*"* name=*"obj"*/>

<%

**int** i=UserDao.register(obj);

**if**(i>0)

out.print("You are successfully registered");

%>

**user.hbm.xml**

<?xml version=*'1.0'* encoding=*'UTF-8'*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"com.javatpoint.mypack.User"* table=*"u400"*>

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"name"*></property>

<property name=*"password"*></property>

<property name=*"email"*></property>

</class>

</hibernate-mapping>

1. **User.java**

**package** com.javatpoint.mypack;

**public** **class** User {

**private** **int** id;

**private** String name, password, email;

**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** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

}

1. **UserDao.java**

**package** com.javatpoint.mypack;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.Configuration;

**public** **class** UserDao {

**public** **static** **int** register(User u) {

**int** i = 0;

Session session = **new** Configuration().configure().buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

t.begin();

i = (Integer) session.save(u);

t.commit();

session.close();

**return** i;

}

}

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">

<!-- Generated by MyEclipse Hibernate Tools. -->

<hibernate-configuration>

<session-factory>

<property name=*"hbm2ddl.auto"*>update</property>

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/office</property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.password"*>pipi</property>

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<mapping resource=*"user.hbm.xml"* />

</session-factory>

</hibernate-configuration>

## Generator classes

The <generator> subelement of id used to generate the unique identifier for the objects of persistent class. There are many generator classes defined in the Hibernate Framework.

All the generator classes implements the **org.hibernate.id.IdentifierGenerator**[**interface**](http://www.javatpoint.com/interface-in-java). The application programmer may create one's own generator classes by implementing the IdentifierGenerator interface. Hibernate framework provides many built-in generator classes:

1. assigned
2. increment
3. sequence
4. hilo
5. native
6. identity
7. seqhilo
8. uuid
9. guid
10. select
11. foreign
12. sequence-identity

### assigned

It is the default generator strategy if there is no <generator> element . In this case, application assigns the id. For example:

....

 <hibernate-mapping>

  <**class** ...>

    <id ...>

     <generator **class**="assigned"></generator>

    </id>

    .....

  </**class**>

 </hibernate-mapping>

### increment

It generates the unique id only if no other process is inserting data into this table. It generates **short**, **int** or **long** type identifier. The first generated identifier is 1 normally and incremented as 1. Syntax:

....

 <hibernate-mapping>

  <**class** ...>

    <id ...>

     <generator **class**="increment"></generator>

    </id>

    .....

  </**class**>

 </hibernate-mapping>

### sequence

It uses the sequence of the database. if there is no sequence defined, it creates a sequence automatically e.g. in case of Oracle database, it creates a sequence named HIBERNATE\_SEQUENCE. In case of Oracle, DB2, SAP DB, Postgre SQL or McKoi, it uses sequence but it uses generator in interbase. Syntax:

.....

 <id ...>

  <generator **class**="sequence"></generator>

 </id>

 .....

For defining your own sequence, use the param subelement of generator.

.....

 <id ...>

  <generator **class**="sequence">

      <param name="sequence">your\_sequence\_name</param>

  </generator>

 </id>

 .....

### 4) hilo

It uses high and low algorithm to generate the id of type short, int and long. Syntax:

1. .....
2. <id ...>
3. <generator **class**="hilo"></generator>
4. </id>
5. .....

### 5) native

It uses identity, sequence or hilo depending on the database vendor. Syntax:

.....

 <id ...>

  <generator **class**="native"></generator>

 </id>

 .....

### 6) identity

It is used in Sybase, My SQL, MS SQL Server, DB2 and HypersonicSQL to support the id column. The returned id is of type short, int or long.

### 7) seqhilo

It uses high and low algorithm on the specified sequence name. The returned id is of type short, int or long.

### 8) uuid

It uses 128-bit UUID algorithm to generate the id. The returned id is of type String, unique within a network (because IP is used). The UUID is represented in hexadecimal digits, 32 in length.

### 9) guid

|  |
| --- |
| It uses GUID generated by database of type string. It works on MS SQL Server and MySQL. |

### 10) select

|  |
| --- |
| It uses the primary key returned by the database trigger. |

### 11) foreign

|  |
| --- |
| It uses the id of another associated object, mostly used with <one-to-one> association. |

### 12) sequence-identity

## Dialects in Hibernate

For connecting any hibernate application with the database, you must specify the SQL dialects. There are many Dialects classes defined for RDBMS in the org.hibernate.dialect package.

|  |  |
| --- | --- |
| RDBMS | Dialect |
| Oracle (any version) | org.hibernate.dialect.OracleDialect |
| Oracle9i | org.hibernate.dialect.Oracle9iDialect |
| Oracle10g | org.hibernate.dialect.Oracle10gDialect |
| MySQL | org.hibernate.dialect.MySQLDialect |
| MySQL with InnoDB | org.hibernate.dialect.MySQLInnoDBDialect |
| MySQL with MyISAM | org.hibernate.dialect.MySQLMyISAMDialect |
| DB2 | org.hibernate.dialect.DB2Dialect |
| DB2 AS/400 | org.hibernate.dialect.DB2400Dialect |
| DB2 OS390 | org.hibernate.dialect.DB2390Dialect |
| Microsoft SQL Server | org.hibernate.dialect.SQLServerDialect |
| Sybase | org.hibernate.dialect.SybaseDialect |
| Sybase Anywhere | org.hibernate.dialect.SybaseAnywhereDialect |
| PostgreSQL | org.hibernate.dialect.PostgreSQLDialect |
| SAP DB | org.hibernate.dialect.SAPDBDialect |
| Informix | org.hibernate.dialect.InformixDialect |
| HypersonicSQL | org.hibernate.dialect.HSQLDialect |
| Ingres | org.hibernate.dialect.IngresDialect |
| Progress | org.hibernate.dialect.ProgressDialect |
| Mckoi SQL | org.hibernate.dialect.MckoiDialect |
| Interbase | org.hibernate.dialect.InterbaseDialect |
| Pointbase | org.hibernate.dialect.PointbaseDialect |
| FrontBase | org.hibernate.dialect.FrontbaseDialect |
| Firebird | org.hibernate.dialect.FirebirdDialect |

## Logging by Log4j

Logging enables the programmer to write the log details into a file permanently. Log4j and Logback frameworks can be used in hibernate framework to support logging.

There are two ways to perform logging using log4j:

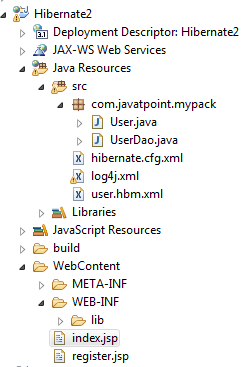
1. By log4j.xml file (or)
2. By log4j.properties file

### Using xml file

Steps to perform Hibernate Logging by Log4j using xml file

There are two ways to perform logging using log4j using xml file:

1. Load the log4j jar files with hibernate
2. Create the log4j.xml file inside the src folder (parallel with hibernate.cfg.xml file)



**Log4j.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j=*"http://jakarta.apache.org/log4j/"*

debug=*"false"*>

<appender name=*"CONSOLE"* class=*"org.apache.log4j.ConsoleAppender"*>

<layout class=*"org.apache.log4j.PatternLayout"*>

<param name=*"ConversionPattern"* value=*"[%d{dd/MM/yy hh:mm:ss:sss z}] %5p %c{2}: %m%n"* />

</layout>

</appender>

<appender name=*"ASYNC"* class=*"org.apache.log4j.AsyncAppender"*>

<appender-ref ref=*"CONSOLE"* />

<appender-ref ref=*"FILE"* />

</appender>

<appender name=*"FILE"* class=*"org.apache.log4j.RollingFileAppender"*>

<param name=*"File"* value=*"C:/projects/Hibernate/javatpointlog.log"* />

<param name=*"MaxBackupIndex"* value=*"100"* />

<layout class=*"org.apache.log4j.PatternLayout"*>

<param name=*"ConversionPattern"* value=*"[%d{dd/MM/yy hh:mm:ss:sss z}] %5p %c{2}: %m%n"* />

</layout>

</appender>

<category name=*"org.hibernate"*>

<priority value=*"DEBUG"* />

</category>

<category name=*"java.sql"*>

<priority value=*"debug"* />

</category>

<root>

<priority value=*"INFO"* />

<appender-ref ref=*"FILE"* />

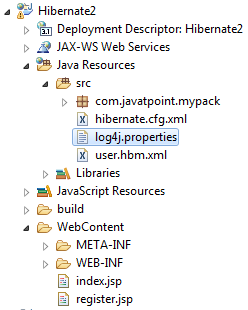
</root>

</log4j:configuration>

### Using properties file

There are two ways to perform logging using log4j using properties file:

1. Load the log4j jar files with hibernate
2. Create the log4j.properties file inside the src folder (parallel with hibernate.cfg.xml file)



**log4j.properties**

# Direct log messages to a log file

log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=C:\\projects\\Hibernate\\javatpointhibernate.log

log4j.appender.file.MaxFileSize=1MB

log4j.appender.file.MaxBackupIndex=1

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c**{1}**:%L - %m%n

# Direct log messages to stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c**{1}**:%L - %m%n

# Root logger option

log4j.rootLogger=INFO, file, stdout

# Log everything. Good for troubleshooting

log4j.logger.org.hibernate=INFO

# Log all JDBC parameters

log4j.logger.org.hibernate.type=ALL

## Inheritance Mapping

We can map the inheritance hierarchy classes with the table of the database.

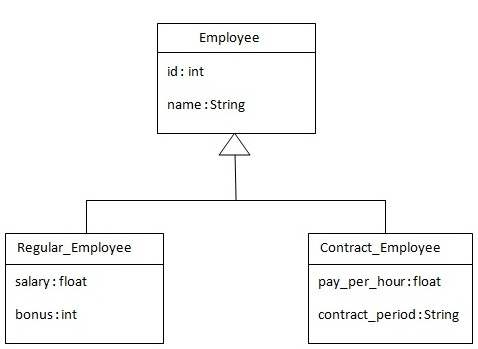
1. Table Per Hierarchy
2. Table Per Concrete class
3. Table Per Subclass

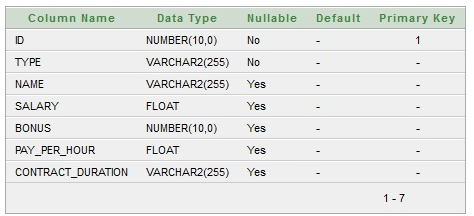
### Table Per Hierarchy

In table per hierarchy mapping, single table is required to map the whole hierarchy, an extra column (known as discriminator column) is added to identify the class. But nullable values are stored in the table .

By this inheritance strategy, we can map the whole hierarchy by single table only. Here, an extra column (also known as **discriminator column**) is created in the table to identify the class.

Let's understand the problem first. I want to map the whole hierarchy given below into one table of the database.





Employee

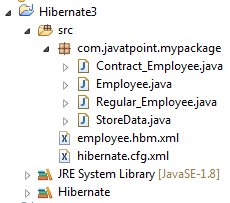
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Id | Salary | bonus | Pay\_per\_hour | Contract\_period | Name |

#### Using xml file

In case of table per class hierarchy an discriminator column is added by the hibernate framework that specifies the type of the record. It is mainly used to distinguish the record. To specify this, **discriminator** subelement of class must be specified.

The **subclass** subelement of class, specifies the subclass. In this case, Regular\_Employee and Contract\_Employee are the subclasses of Employee class.

1. Create new project add required jars



1. Create bean classes

**package** com.javatpoint.mypackage;

**public** **class** Employee {

**private** **int** id;

**private** String name;

**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;

}

}

**package** com.javatpoint.mypackage;

**public** **class** Contract\_Employee **extends** Employee {

**private** **float** pay\_per\_hour;

**private** String contract\_duration;

**public** **float** getPay\_per\_hour() {

**return** pay\_per\_hour;

}

**public** **void** setPay\_per\_hour(**float** pay\_per\_hour) {

**this**.pay\_per\_hour = pay\_per\_hour;

}

**public** String getContract\_duration() {

**return** contract\_duration;

}

**public** **void** setContract\_duration(String contract\_duration) {

**this**.contract\_duration = contract\_duration;

}

}

**package** com.javatpoint.mypackage;

**public** **class** Regular\_Employee **extends** Employee {

**private** **float** salary;

**private** **int** bonus;

**public** **float** getSalary() {

**return** salary;

}

**public** **void** setSalary(**float** salary) {

**this**.salary = salary;

}

**public** **int** getBonus() {

**return** bonus;

}

**public** **void** setBonus(**int** bonus) {

**this**.bonus = bonus;

}

}

1. employee.hbm.xml

<?xml version=*'1.0'* encoding=*'UTF-8'*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"com.javatpoint.mypackage.Employee"* table=*"emp121"*

discriminator-value=*"emp"*>

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

<discriminator column=*"type"* type=*"string"*></discriminator>

<property name=*"name"*></property>

<subclass name=*"com.javatpoint.mypackage.Regular\_Employee"*

discriminator-value=*"reg\_emp"*>

<property name=*"salary"*></property>

<property name=*"bonus"*></property>

</subclass>

<subclass name=*"com.javatpoint.mypackage.Contract\_Employee"*

discriminator-value=*"con\_emp"*>

<property name=*"pay\_per\_hour"*></property>

<property name=*"contract\_duration"*></property>

</subclass>

</class>

</hibernate-mapping>

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>

<property name=*"hbm2ddl.auto"*>update</property>

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/office</property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.password"*>pipi</property>

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<mapping resource=*"employee.hbm.xml"* />

</session-factory>

</hibernate-configuration>

1. StoreData.java

**package** com.javatpoint.mypackage;

**import** org.hibernate.\*;

**import** org.hibernate.cfg.\*;

**public** **class** StoreData {

**public** **static** **void** main(String[] args) {

Session session = **new** Configuration().configure("hibernate.cfg.xml").buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

Employee e1 = **new** Employee();

e1.setName("sonoo");

Regular\_Employee e2 = **new** Regular\_Employee();

e2.setName("Vivek Kumar");

e2.setSalary(50000);

e2.setBonus(5);

Contract\_Employee e3 = **new** Contract\_Employee();

e3.setName("Arjun Kumar");

e3.setPay\_per\_hour(1000);

e3.setContract\_duration("15 hours");

session.persist(e1);

session.persist(e2);

session.persist(e3);

t.commit();

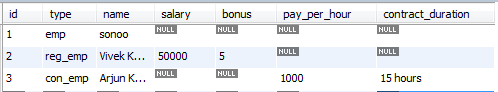
session.close();

System.***out***.println("success");

}

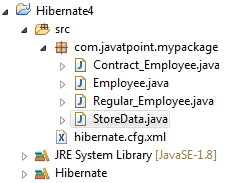
}

Output:



#### using Annotation

1. create new project



1. create bean classes

**package** com.javatpoint.mypackage;

**import** javax.persistence.Column;

**import** javax.persistence.DiscriminatorColumn;

**import** javax.persistence.DiscriminatorType;

**import** javax.persistence.DiscriminatorValue;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Inheritance;

**import** javax.persistence.InheritanceType;

**import** javax.persistence.Table;

@Entity

@Table(name = "emp01")

@Inheritance(strategy = InheritanceType.***SINGLE\_TABLE***)

@DiscriminatorColumn(name = "type", discriminatorType = DiscriminatorType.***STRING***)

@DiscriminatorValue(value = "employee")

**public** **class** Employee {

@Id

@GeneratedValue(strategy=GenerationType.***AUTO***)

@Column(name="id")

**private** **int** id;

@Column(name="name")

**private** String name;

**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;

}

}

**package** com.javatpoint.mypackage;

**import** javax.persistence.Column;

**import** javax.persistence.DiscriminatorValue;

**import** javax.persistence.Entity;

@Entity

@DiscriminatorValue("contractemployee")

**public** **class** Contract\_Employee **extends** Employee {

@Column(name = "pay\_per\_hour")

**private** **float** pay\_per\_hour;

@Column(name = "contract\_duration")

**private** String contract\_duration;

**public** **float** getPay\_per\_hour() {

**return** pay\_per\_hour;

}

**public** **void** setPay\_per\_hour(**float** pay\_per\_hour) {

**this**.pay\_per\_hour = pay\_per\_hour;

}

**public** String getContract\_duration() {

**return** contract\_duration;

}

**public** **void** setContract\_duration(String contract\_duration) {

**this**.contract\_duration = contract\_duration;

}

}

**package** com.javatpoint.mypackage;

**import** javax.persistence.Column;

**import** javax.persistence.DiscriminatorValue;

**import** javax.persistence.Entity;

@Entity

@DiscriminatorValue("regularemployee")

**public** **class** Regular\_Employee **extends** Employee {

@Column(name = "salary")

**private** **float** salary;

@Column(name = "bonus")

**private** **int** bonus;

**public** **float** getSalary() {

**return** salary;

}

**public** **void** setSalary(**float** salary) {

**this**.salary = salary;

}

**public** **int** getBonus() {

**return** bonus;

}

**public** **void** setBonus(**int** bonus) {

**this**.bonus = bonus;

}

}

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>

<property name=*"hbm2ddl.auto"*>update</property>

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/office</property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.password"*>pipi</property>

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<mapping class=*"com.javatpoint.mypackage.Employee"* />

<mapping class=*"com.javatpoint.mypackage.Contract\_Employee"* />

<mapping class=*"com.javatpoint.mypackage.Regular\_Employee"* />

</session-factory>

</hibernate-configuration>

StoreDate.java

**package** com.javatpoint.mypackage;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**public** **class** StoreData {

**public** **static** **void** main(String[] args) {

AnnotationConfiguration configuration = **new** AnnotationConfiguration();

Session session = configuration.configure("hibernate.cfg.xml").buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

Employee e1 = **new** Employee();

e1.setName("Sama");

Regular\_Employee e2 = **new** Regular\_Employee();

e2.setName("Mala");

e2.setSalary(50000);

e2.setBonus(5);

Contract\_Employee e3 = **new** Contract\_Employee();

e3.setName("Amara");

e3.setPay\_per\_hour(1000);

e3.setContract\_duration("15 hours");

session.persist(e1);

session.persist(e2);

session.persist(e3);

t.commit();

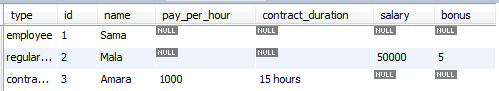
session.close();

System.***out***.println("success");

}

}

output:



You need to use @Inheritance(strategy=InheritanceType.SINGLE\_TABLE), @DiscriminatorColumn and @DiscriminatorValue annotations for mapping table per hierarchy strategy.

Employee

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Id | Salary | bonus | Pay\_per\_hour | Contract\_period | Name |

|  |
| --- |
|  |
|  |

Employee

|  |  |
| --- | --- |
| Id | Name |

Regular\_Employee

|  |  |  |
| --- | --- | --- |
| Id | Salary | bonus |

Contract\_Employee

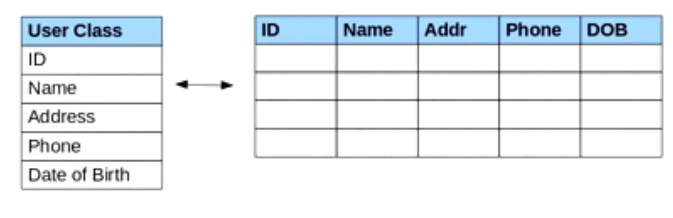
|  |  |  |
| --- | --- | --- |
| Id | Pay\_per\_hour | Contract\_period |

# Hibernate (javabrains)

## Introduction

* An ORM (Object Relational Mapping ) tool.
* Used in the data layer of applications
* Implements JPA(Java Persistence API ).

## The problem



* Mapping member variables to columns.
* Mapping relationships.
* Handling data types.
* Managing changes to object state.

Solution is 🡪

Object Relational Mapping

## Install Hibernate

E-clips-

1. Go to 🡪 [www.hibernate.org/downloads](http://www.hibernate.org/downloads) and download stable version
2. Open e-clips 🡪 new Project 🡪 give a name 🡪 next 🡪 Finish
3. Write click the project 🡪 properties 🡪 java build path 🡪 Add library 🡪 user Libraries 🡪 new 🡪 give a name 🡪 Add jars 🡪

* Hibernate distribution 🡪 Hibernate3.jar
* Lib 🡪 required 🡪 add all .jar ‘s
* Lib 🡪 jpa 🡪 add the .jar
* Lib 🡪 bytecode 🡪 select one .jar from two of them you like

Netbeans –

1. new project 🡪 give project name 🡪 next 🡪 select Hibernate 🡪 finish

After that you need add **database diver.**

## Using Hibernate

### Without hibernate

* JDBC Database configuration (IP, Port number, user, password).
* The model object (user object etc.).
* Service method to create the model object.
* Database design.
* DOA method to save the object using SQL queries.

### With hibernate

* JDBC Database configuration. – hibernate
* The model object – annotations
* Service method to create the model object – use the Hibernate API
* Database design. – not need
* DOA method to save the object using SQL queries. – not need

### JDBC Database configuration

Got to

Src 🡪 hibernate.cfghibernate.cfg.xml

### The model object – annotations

package models;

import javax.persistence.Entity;

import javax.persistence.Id;

// this annotation mean hibernate treat this clas as a Entity

@Entity

public class **Test** {

// this annotation describes the "private int id;" will be the primary key

@Id

private int id;

private String name;

private String address;

/\*\*

\* **@return** the id

\*/

public int **getId**() {

return id;

}

public **Test**(int id, String name, String address) {

this.id = id;

this.name = name;

this.address = address;

}

/\*\*

\* **@return** the name

\*/

public String **getName**() {

return name;

}

/\*\*

\* **@return** the address

\*/

public String **getAddress**() {

return address;

}

/\*\*

\* **@param** id the id to set

\*/

public void **setId**(int id) {

this.id = id;

}

/\*\*

\* **@param** name the name to set

\*/

public void **setName**(String name) {

this.name = name;

}

/\*\*

\* **@param** address the address to set

\*/

public void **setAddress**(String address) {

this.address = address;

}

}

Hibernate.cfg.xml

<!-- Name the anotated entity class -->

<mapping class="models.Test" />

## Using the hibernate API

1. **Create a session factory** – Is a one object which creates sessions depending on how many session we want throughout the application (in it’s execution period).

If save sonting – get the session factory and that session is use to save.

hibernate.cfghibernate.cfg.xml is use to tell what are databases, what are the configurations we need.

1. **Create session using the session factry –**
2. Use the session to save model object.