Introduction to Hibernate

Agenda

1. Your first Hibernate program
2. Hibernate Architecture
3. Hibernate APIs

Introduction to Hibernate

- What is Hibernate?
  - Java Framework at database layer of an application
  - It's an ORM tool
  - Implements JPA (Java Persistence API) Specification

- Why Hibernate?
  - It maps the Java Classes to Database Tables
  - For simple applications Database design is not required
    - Using hibernate we can create tables structures and relations based on Java classes
  - Persisting of Java Objects to Database is done with simple APIs
  - Smart fetching Strategies to minimize database fetching connects
  - Powerful yet Simple Querying mechanism

A simple Hibernate program

Let us look at an example and get the feel of Hibernate before understanding the underlying concepts

- Let us use the DEPT table under SCOTT schema in Oracle SQL Plus

```
SQL> desc dept
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 DEPTNO                                             NUMBER(3)
 DNAME                                              VARCHAR2(14)
 LOC                                                VARCHAR2(13)

SQL> select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

A simple Hibernate program

- Let us Create a Maven Project and insert a new record into the DEPT table using Hibernate
- How to create a maven Project , Setting up the dependencies step by step guide is given in the below link

For Sample Demo Refer : Hibernate Demo1.pdf

- There are 4 simple things we need to observe here
  1. The Bean class which represents the table
  2. The Hibernate mapping file which maps the bean class to the table
  3. The Hibernate configuration file which gives the basic DB configuration details
  4. Finally, the main program which performs the insert into the database table

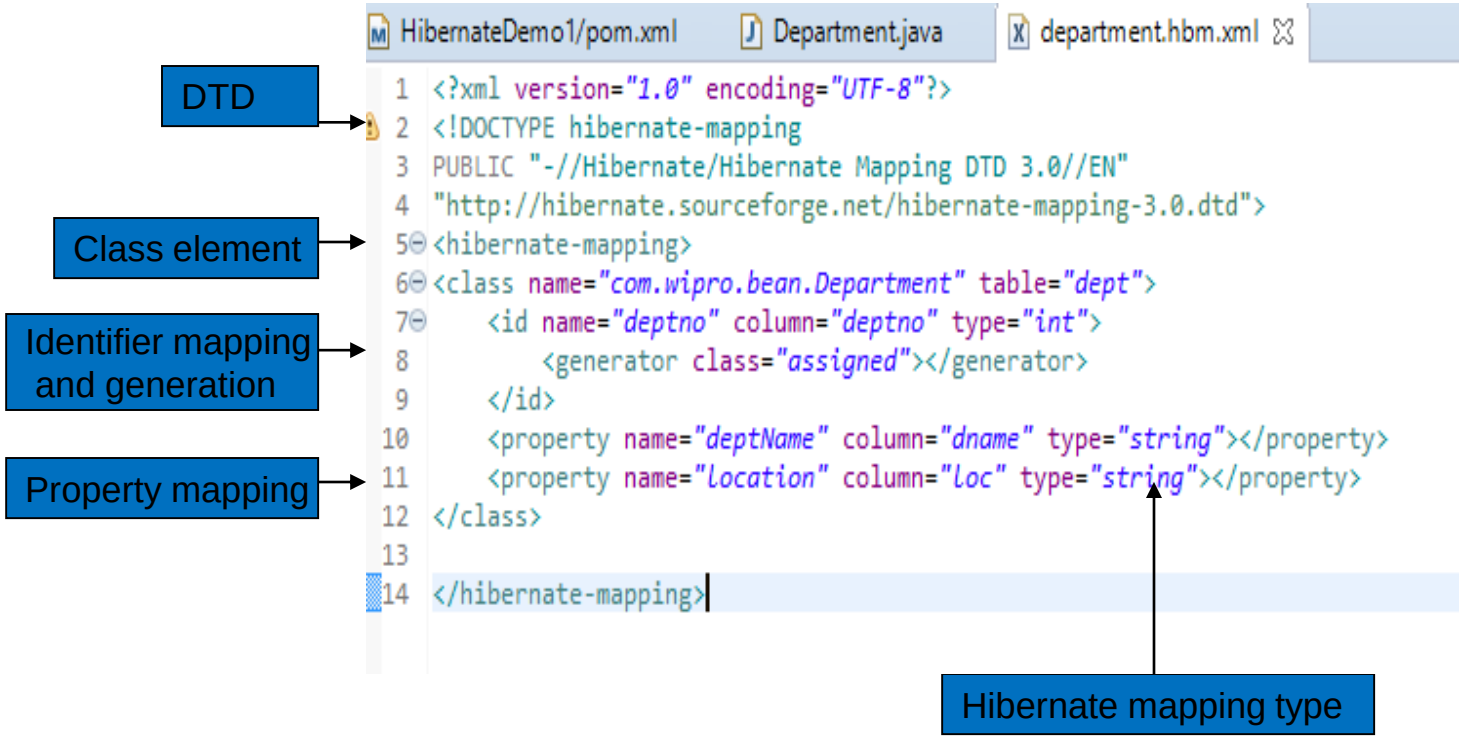
A simple Hibernate program (Contd.). Bean Class

- In the example a POJO (plain old Java object) is used to represent the data in the table
- The Java class specifies the fields/attributes as variables to represent the columns in the table
- It has setter (mutator) and getter (accessor) methods for each of the properties in the class
- It uses these simple setters and getters to retrieve or write the data
- The class properties will be mapped to columns in the mapping file
- It has a no-argument constructor (mandatory), as Hibernate uses reflection to create instances

```
public class Department {
private int deptno;
private String deptName;
private String location;
public Department() {
//mandate no argument constructor
}
public Department(int deptno, String
deptName, String location) {
super();
this.deptno = deptno;
this.deptName = deptName;
this.location = location;
}
public int getDeptno() {
return deptno;
}
public void setDeptno(int deptno) {
this.deptno = deptno;
} … }
```

A simple Hibernate program (Contd.). : Mapping File

Create an XML hibernate mapping definition file department.hbm.xml



A simple Hibernate program (Contd.). : Mapping File

- Mapping POJOs to tables can be done using a Hibernate mapping file or Annotations
- Object/relational mappings defined using mapping file is an XML document
- The mapping language is Java-centric, i.e. mappings are constructed around persistent class declarations, and not table declarations
- Mapping file tells Hibernate which classes and attributes are mapped to which tables and columns for the use of loading and storing
- The Department objects are called “Persistent Objects” or “Entities” for they can be persisted in database and they represent real world entities
- The naming convention for mapping files is to use the name of the persistent class with the **hbm.xml** extension
- The mapping file for the Department class is **Department.hbm.xml**

A simple Hibernate program (Contd.). : Mapping File

- class tag : <class name="com.wipro.bean.Department" table="dept">
  - name attribute - requires the name of the POJO class
  - table attribute - requires the name of the corresponding table in the database
- id tag : <id name="deptno" column="deptno" type="int">
  - Each persistent object must have an identifier
  - It is used by Hibernate to identify that object uniquely
  - The id element also describes how the key value is generated : <generator class="assigned"></generator>
  - Here we have used class value assigned meaning data is passed by the application or user
  - Some predefined generator class values are : increment , sequence , identity etc.
- property tags : <property name="deptName" column="dname" type="string"></property>
  - provides names of the properties of the class, their data types and the name of the column to which the property should map onto

A simple Hibernate program (Contd.). : Configuration file

- Create an XML configuration file hibernate.cfg.xml which provides Hibernate with all the details needed to connect to the database

```
<hibernate-configuration>
<session-factory>
<!-- Oracle Dialect -->
<property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
<!-- Database Connection Settings -->
<property name="hibernate.connection.driver_class">oracle.jdbc.OracleDriver</property>
<property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
<property name="hibernate.connection.username">scott</property>
<property name="hibernate.connection.password">tiger</property>
<!-- To echo / show all executed queries on server output -->
<property name="hibernate.show_sql">true</property>
<!-- Used to Create or Alter Table  -->
<!– When the Given class structure not present in the database -->
<property name="hibernate.bhm2ddl.auto">update</property>
<!-- Mapping file entry -->
<mapping resource="department.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

A simple Hibernate program (Contd.).

- All configuration details are given through the XML file
- the default xml configuration file is called **hibernate.cfg.xml**
- It is placed directly in the source directory and not inside any package
  - *In **Maven** it is placed under the **src/main/resources***
- The configuration file provides details of the database settings and details about the connection pooling class, logging and mapping files

```
<!-- JDBC connection pool -->
<property name="connection.pool_size">1</property>
```

- hibernate.dialect : provides the classname to Hibernate in order to generate optimized SQL for a particular relational database

A simple Hibernate program (Contd.).

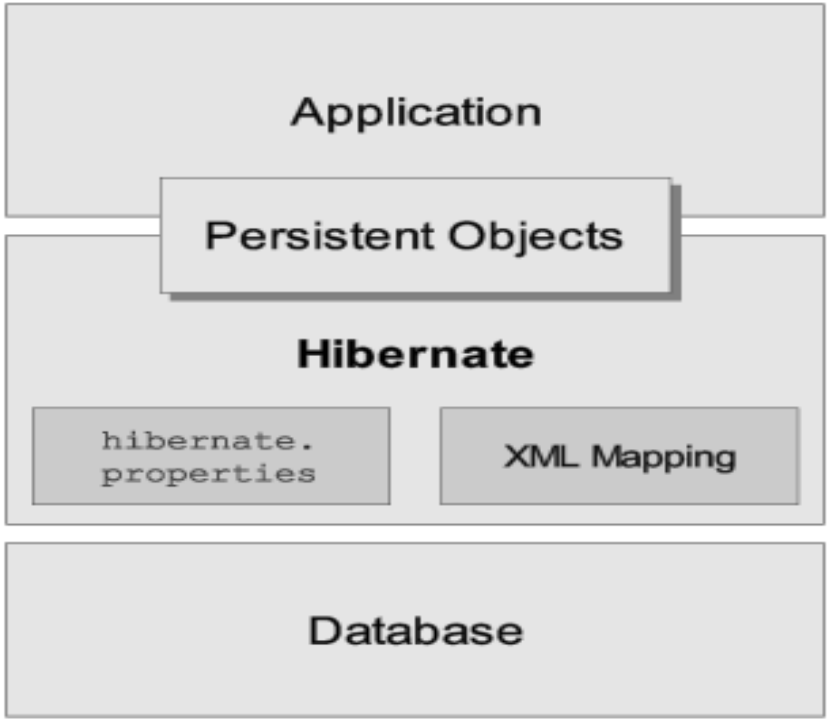
- In the main class we can see
  - There was no connection created or explicit SQL queries
  - Instead there was *Configuration* object, *SessionFactory* Object And *Session* Object
  - Which Inserted the Department class object to the DB using simple **save** method

```
public static void main(String[] args) {
Configuration cfg = new Configuration().configure();
SessionFactory sf = cfg.buildSessionFactory();
Session session = sf.openSession();
Transaction transaction = session.beginTransaction();
Department deptobj = new Department(50, "TT", "NJ");
session.save(deptobj);
transaction.commit();
System.out.println("Record inserted");
System.out.println(deptobj);
session.close();
}
```

- To understand how Insert worked we need to understand the ***Hibernate Architecture***

Hibernate Architecture

Now let us understand the Hibernate architecture

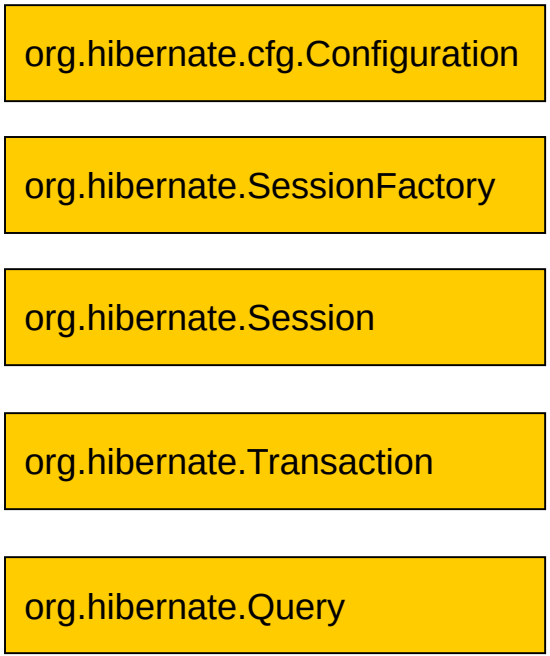


Hibernate uses persistent objects (POJOs) along with XML mapping documents/Annotations for persisting objects to the database layer

Hibernate uses database and configuration data to provide persistence services (and persistent objects) to the application

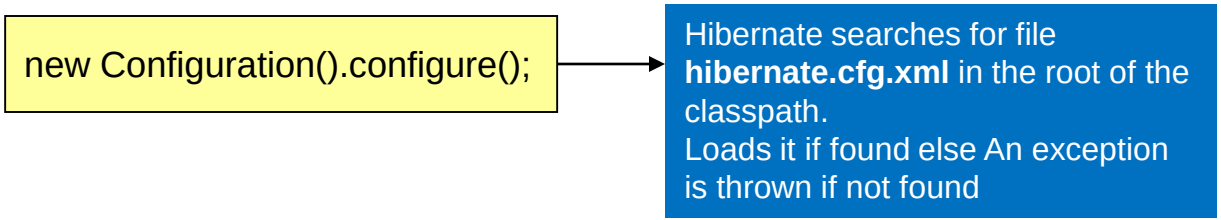
Hibernate API

- Some mandatory APIs used for performing CRUD operations on to the database



Hibernate API – The Configuration class

- Hibernate configuration
  - The Configuration object needs to know about all XML mapping files before you build the SessionFactory
  - Represents a set of mapping files specified in the Hibernate configuration file



Hibernate API – The Configuration File

- All the configuration information is stored in an xml file named hibernate.cfg.xml
- Alternatively the configuration information can also be stored in hibernate.properties file
- Different categories of information is stored in the configuration file
- Following are few of them:
  - Database dialect and connection
  - Database driver (Driver class, Driver URL)
  - Connection pool
  - Caching
  - Transaction
  - Logging

Hibernate API – The Configuration File

- Database Dialect
  - Based on your requirement you use different Databases like MySQL, PostgreSQL, Oracle etc.
  - Hibernate supports a lot of dialects for different databases and their corresponding versions to generate Optimized SQL
  - Hibernate can also choose the correct Dialect implementation based on the JDBC metadata
- Connection information
  - Database Driver class, URL, Username and Password
- Connection Pool is provided by default
- we can still use a third party utility for a production environment.
  - Like C3P0, Proxool connection pooling facility.

Hibernate API – The Configuration File

- Caching
  - 2 levels of caching is provided for better performance
  - we can switch on or switch off the second level cache in the configuration file
- Transaction
  - We can set the transaction contexts like JTA, JDBC
- Logging
  - Hibernate uses log4j for logging, which can be enabled here
- Data Source
  - JNDI information for connecting to a Database through a managed environment is provided here

Hibernate API – The SessionFactory Interface

- Initialize Hibernate by building a SessionFactory object from a Configuration object
- A single SessionFactory object represents a particular Hibernate database setting configuration for a particular Database.
- An Hibernate startup procedure using automatic(default : hibernate.cfg.xml) configuration file detection:

```
SessionFactory sessionFactory = new
Configuration().configure().buildSessionFactory();
```

- The SessionFactory is thread-safe and can be shared among application threads

Hibernate API – The Session Interface

- A Session is created using SessionFactory instance

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

- Factory for Transaction – specifies transaction boundaries

```
Transaction tx = session.beginTransaction();
```

- Referred as a unit of work
  - To begin a unit of work you open a Session
  - To end a unit of work you close a Session
- Session Objects are Responsible for storing and retrieving objects

```
session.save( a persistence object );
```

Summary

In this module you will be able to:

- Write a Simple Hibernate program
- Understand the Architecture
- Use Hibernate API

Thank You