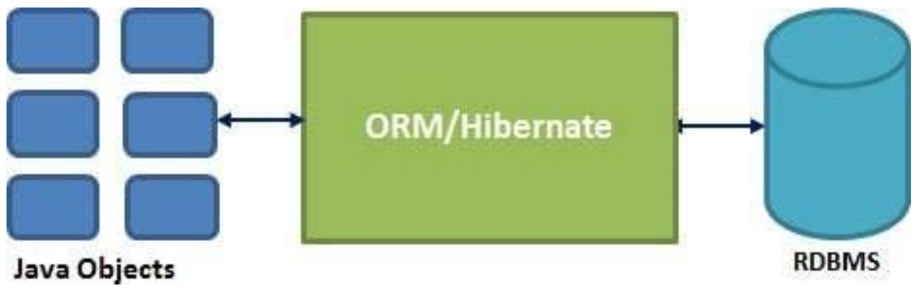
- **Introduction To Hibernate**

Hibernate is an **O**bject-**R**elational **M**apping (ORM) solution for JAVA. It is an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieves the developer from 95% of common data persistence related programming tasks.

Hibernate sits between traditional Java objects and database server to handle all the works in persisting those objects based on the appropriate O/R mechanisms and patterns.



Hibernate Advantages

- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
- Provides simple APIs for storing and retrieving Java objects directly to and from the database.
- If there is change in the database or in any table, then you need to change the XML file properties only.
- Abstracts away the unfamiliar SQL types and provides a way to work around familiar Java Objects.
- Hibernate does not require an application server to operate.
- Manipulates Complex associations of objects of your database.
- Minimizes database access with smart fetching strategies.
- Provides simple querying of data.

Supported Databases

Hibernate supports almost all the major RDBMS. Following is a list of few of the database engines supported by Hibernate −

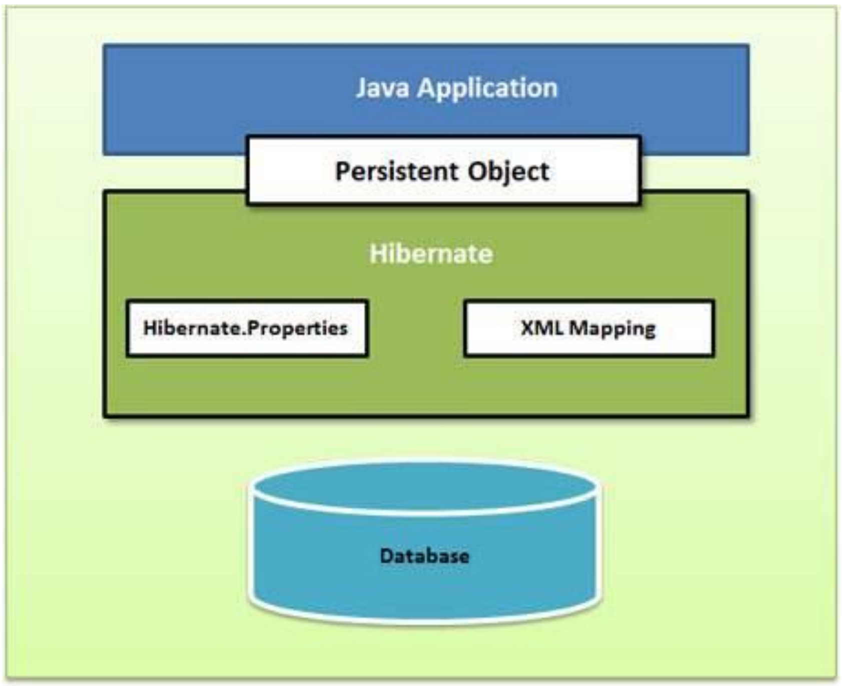
- HSQL Database Engine
- DB2/NT
- MySQL

- PostgreSQL
- FrontBase
- Oracle
- Microsoft SQL Server Database
- Sybase SQL Server
- Informix Dynamic Server

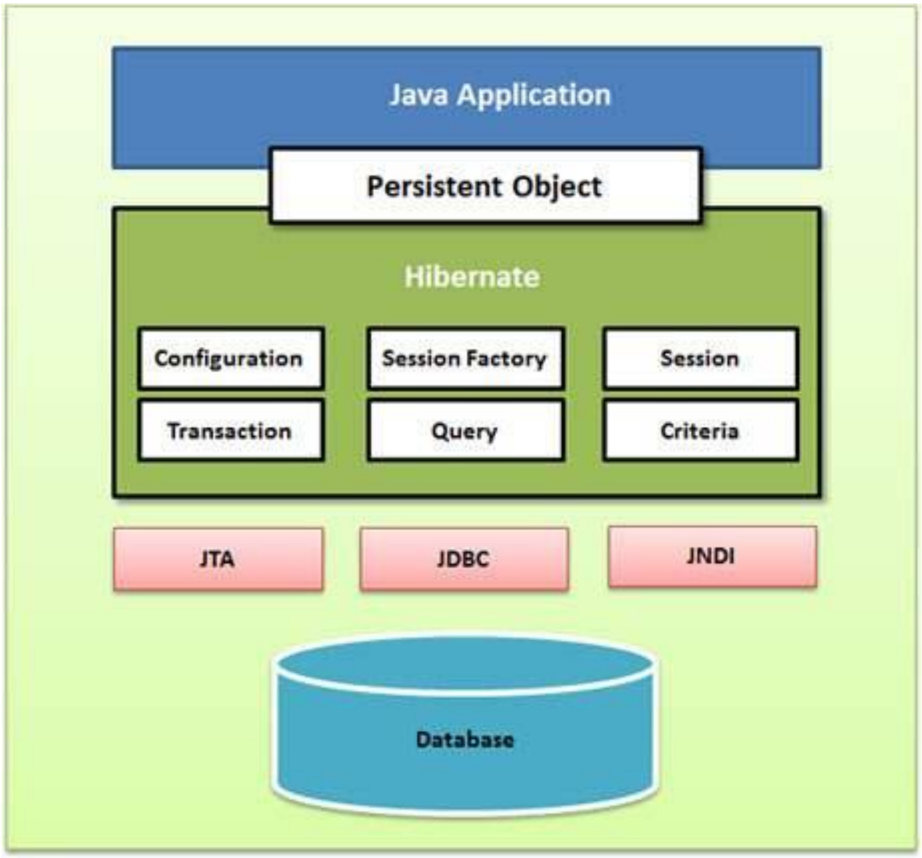
Hibernate Architecture

Hibernate has a layered architecture which helps the user to operate without having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.

Following is a very high level view of the Hibernate Application Architecture.



Following is a detailed view of the Hibernate Application Architecture with its important core classes.



Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a rudimentary level of abstraction of functionality common to relational databases, allowing almost any database with a JDBC driver to be supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers.

Following section gives brief description of each of the class objects involved in Hibernate Application Architecture.

1)Configuration Object

The Configuration object is the first Hibernate object you create in any Hibernate application. It is usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate.

The Configuration object provides two keys components −

- **Database Connection** − This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.
- **Class Mapping Setup** − This component creates the connection between the Java classes and database tables.

2)SessionFactory Object

Configuration object is used to create a SessionFactory object which in turn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application.

The SessionFactory is a heavyweight object; it is usually created during application start up and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So, if you are using multiple databases, then you would have to create multiple SessionFactory objects.

3)Session Object

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

4)Transaction Object

A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

5)Query Object

Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

6)Criteria Object

Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

Creating Simple Hibernate Application

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** .

2) Add hibernate capabilities

To add the jar files **select your project - click on MyEclipse menu - Project Capabilities - add Hibernate capabilities- next- next**.

3) Create the Persistent class

Here, we are creating the same persistent class which we have created in the previous topic. To create the persistent class, Right click on src - New - Class - specify the class with package name - finish.

Employee.java

```
package com.hibernate.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;
}
}
```

4) Create the mapping file for Persistent class

Here, we are creating the same mapping file as created in the previous topic. 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 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
 <hibernate-mapping>
  <class name="com.hibernate.mypackage.Employee" table="emp1000">
    <id name="id">
     <generator class="assigned"></generator>
    </id>
    <property name="firstName"></property>
    <property name="lastName"></property>
  </class>
 </hibernate-mapping>
```

5) Add mapping of hbm file in configuration file

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

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

Now the configuration file will look like this:

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.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>
```

6) Create the class that retrieves or stores the persistent object

```
package com.hibernate.mypackage;  // employee object to the database
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(115);
    e1.setFirstName("sonoo");
    e1.setLastName("jaiswal");
    session.persist(e1);//persisting the object
    t.commit();//transaction is commited
    session.close();
    System.out.println("successfully saved");
}
}
```

7) Add the jar file for oracle (ojdbc14.jar)

To add the ojdbc14.jar file, right click on your project - build path - add external archives - select the ojdbc14.jar file - open.

8) Run the application