HIBERNATE

Objective…

- To understand the Hibernate Technology and their usage and role with the persistance layer
- To understand the mapping between a POJO class and the set of tables underlined in the persistance layer
- To gain a good understanding about the implementation of hibernate framework
- To understand the various possible mappings
- To understand HQL (Hibernate Query Language)

Agenda… Day 1

- *ORM tools compared with traditional JDBC tools*
  - o ORM benefits
  - o The need for Hibernate
- *Introduction to Hibernate*
  - o Hibernate Architecture
  - o First Hibernate application
- *Hibernate Configurations and Sessions*
  - o Session factory and sessions
- *Mapping a POJO to a table*
  - o Mapping files and the class element
  - o Identifier fields
  - o Property mappings
  - o Hibernate data types
- *Association Mapping*
  - o Unidirectional mapping without join tables
- *Working with data from the Database*
  - o Loading data
  - o Updating data
  - o Deleting data

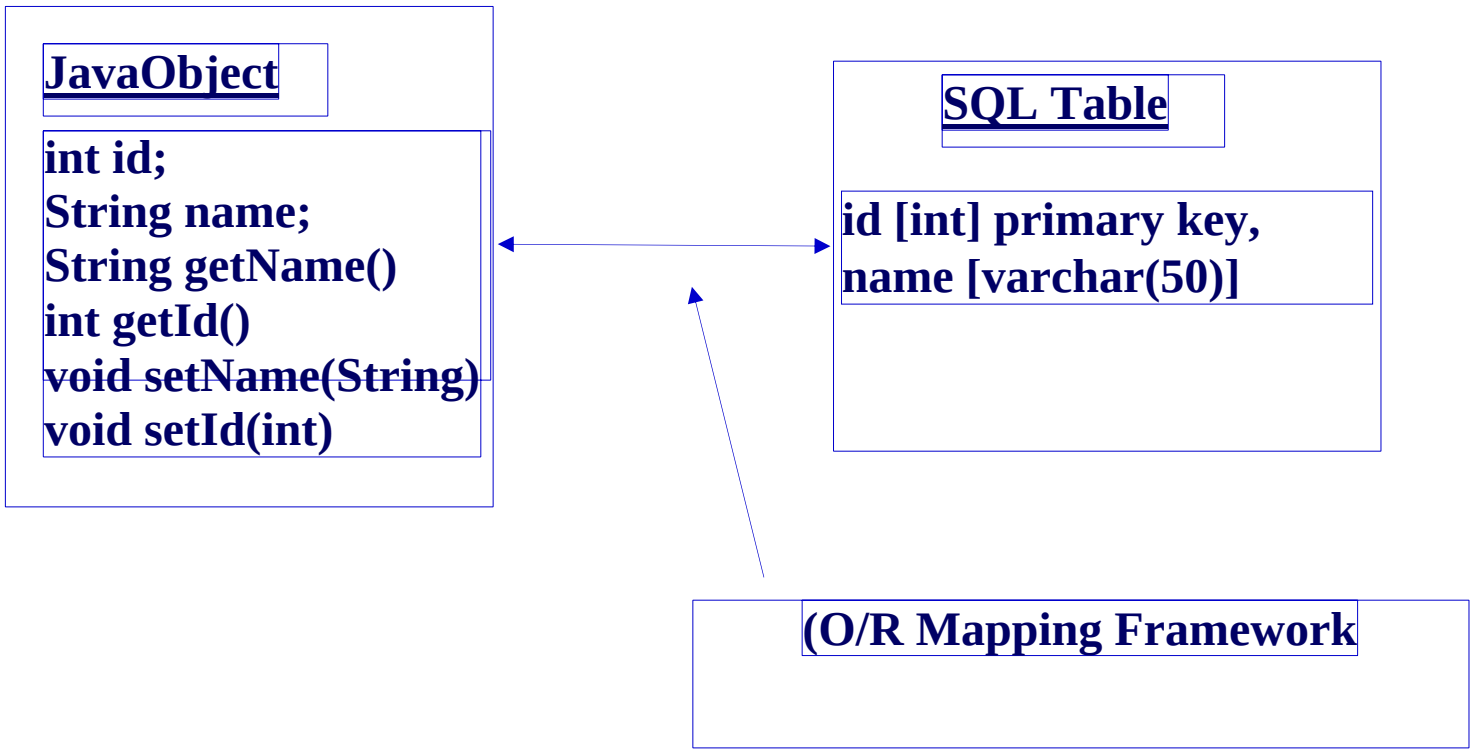
Agenda… Day 2 (half a day)

- *Querying data*
- *HQL*
  - Select clause
  - Narrowing with a where clauses
  - Order by
- Using Native SQL

Traditional JDBC Tools

- The program is exposed the underlying Datasource or Driver Manager.
- Format mismatch between the objects and database.
- The application is not all java
- Configuration file has to be created in the file system, usually a property text file.
- Even if a property file is used, the way in which the configuration is stored is not following any standard (like xml)
- Working with both OO software and Relational Databases can be cumbersome and time consuming.

ORM-Object Relational Mapping



ORM Advantages

- Abstraction of the underlying Data Source or DriverManager from the program
- All Java and All Object oriented program
- Configuration files used follow standard (xml-dtd)
- All configuration are outside the application, so changes will not affect the application
- Object Oriented Query Language (No two different formats)

What is Hibernate?

- An *open source tool* for connecting to database
- It is a ORM tool – Object Relational Mapping Tool for Java
- It replaces direct persistence related database access with hith-level object handling functions
- Solves the problems related with Object-Relational mismatch

HIBERNATE

- Base for EJB 3.0
- Act as an interface between your java program and Database engine
- Mapping Java data types to SQL data types
- Provides data query and retrieval facilities
- Abstract away unfamiliar SQL types and provide programmer to work around familiar Java Objects..
- Provides Simple querying of data with HQL.
- Pluggable to many famous database engines.

Database engines supported

- Java to the Database dialect conversion is taken care by the Hibernate frame work and it supports the following databases
  - Oracle
  - DB2
  - Microsoft SQL Server
  - Sybase
  - MySQL
  - HSQL Database Engine
  - Informix
  - And more

Advantages of Hibernate

- One of the best Model tool for MVC Architecture
- Increased Developer productivity and reduced maintenance costs
- Standard driven development
- Optimised performance
- Flexible and simple APIs
- High performance – straight JDBC Coding

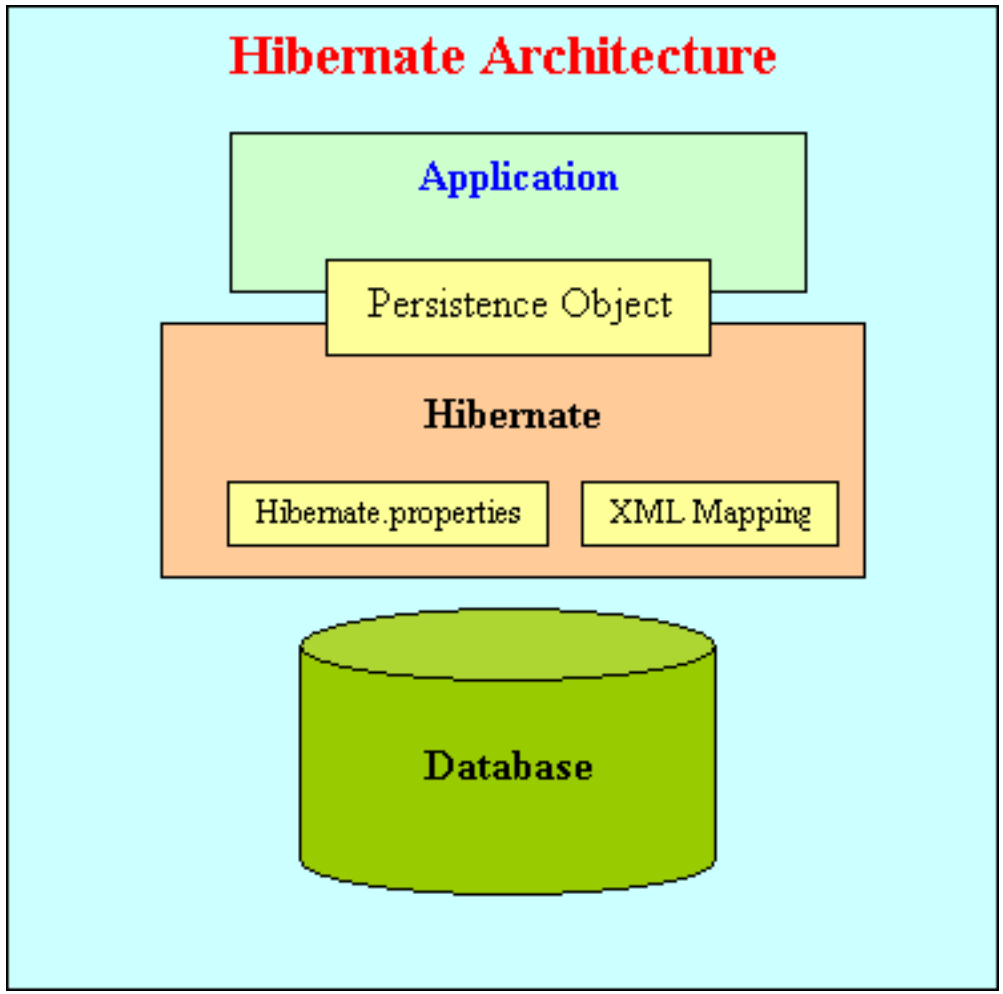
Hibernate Features

- Hibernate 3.0 provides three full-featured query facilities: Hibernate Query Language, the newly enhanced Hibernate Criteria Query API, and enhanced support for queries expressed in the native SQL dialect of the database.
- Filters for working with temporal (historical), regional or permissioned data.
- Enhanced Criteria query API: with full support for projection/aggregation and subselects.
- Runtime performance monitoring: via JMX or local Java API, including a second-level cache browser

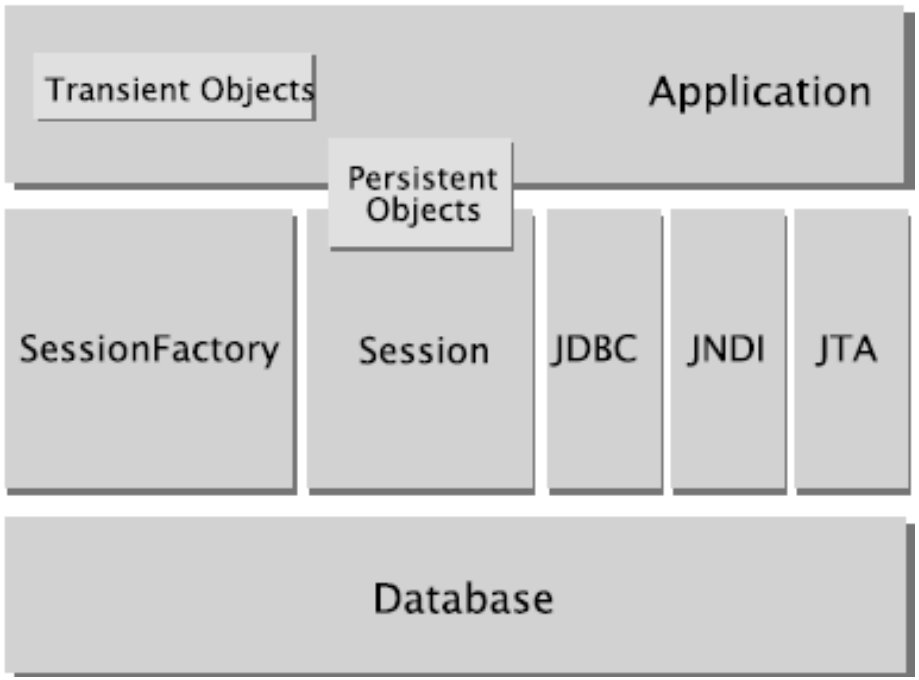
Hibernate Features

- Hibernate is Free under LGPL: Hibernate can be used to develop/package and distribute the applications for free.
- Hibernate is Scalable: Hibernate is very performant and due to its dual-layer architecture can be used in the clustered environments.
- Less Development Time: Hibernate reduces the development timings as it supports inheritance, polymorphism, composition and the Java Collection framework.

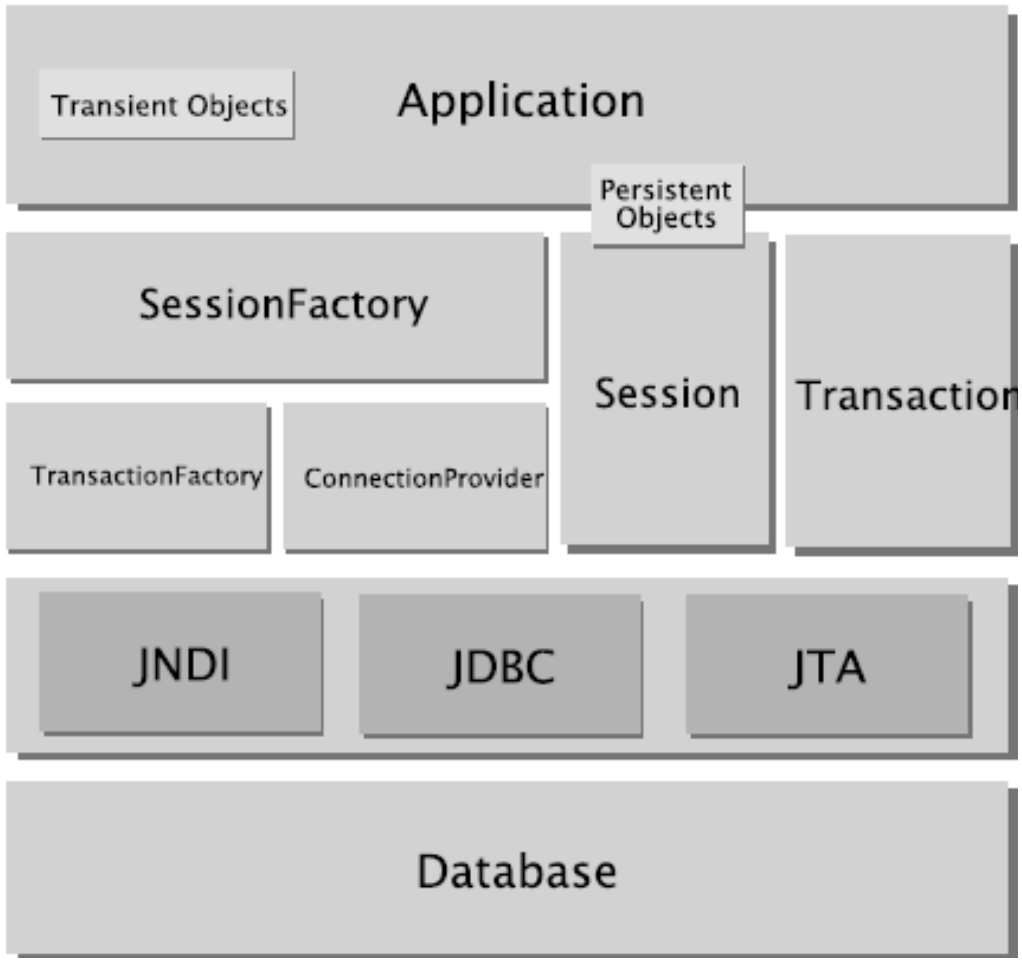
Hibernate Architecture Level 1



Hibernate Architecture Level 2



Hibernate Architecture Level 3



Hibernate Components

- **Hibernate architecture has three main components:**
  - **Connection Management**
  - **Transaction Management**
  - **Object relational Mapping**

Persistant Objects

Session Factory

Session

Transaction Factory

Transaction

ConnectionProvider

Hibernate Components

- Persistant Objects
- Session Factory
- Session
- Transaction Factory
- Transaction
- ConnectionProvider

Persistant Objects

- **Persistant Object**

  Short-lived, single threaded objects containing persistent state and business function. These might be ordinary JavaBeans/POJOs, the only special thing about them is that they are currently associated with (exactly one) `Session`.

- **Persistant Object** are of two types

  Transient Objects

  Detached Objects

Transient/Detached Objects

Transient Object:

Instance of a non persistent class

Detached Object:

Instances of persistent classes without any session association.

The objects in persistent and bound to a hibernate session will
Undergo Automatic dirty checking
This is done by automatically by hibernate collection of objects

Session Factory and Session

SessionFactory (`org.hibernate.SessionFactory`)

A threadsafe (immutable) cache of compiled mappings for a single database. A factory for `Session` and a client of `Connection Provider`.

Session (`org.hibernate.Session`)

A single-threaded, short-lived object representing a conversation between the application and the persistent store. Wraps a JDBC connection. Factory for `Transaction`. Holds a mandatory (first-level) cache of persistent objects, used when navigating the object graph or looking up objects by identifier. Session internals consist of

a. **A queue** SQL statements that need to be synchronized with the database at some point

b. **A map** of managed persistence instances that are monitored by the Session

Transaction Factory and Transaction

TransactionFactory (`org.hibernate.TransactionFactory`)

A factory for `Transaction` instances. Not exposed to the application. (Optional)

Transaction (`org.hibernate.Transaction`)

A single-threaded, short-lived object used by the application to specify atomic units of work. (Optional)

ConnectionProvider

ConnectionProvider (`org.hibernate.connection.ConnectionProvider`)

A factory for (and pool of) JDBC connections. Abstracts application from underlying `Datasource` or `DriverManager`. Not exposed to application. (Optional)

First Hibernate Application

- The minimum set of Jars required
  - antlr.jar
  - cglib.jar
  - asm.jar
  - asm-attrs.jars
  - commons-collections.jar
  - commons-logging.jar
  - hibernate3.jar
  - jta.jar
  - dom4j.jar
  - log4j.jar

The persistant Pojo Class

- Plain Old Java Object
- A simple Java bean representing one row of the table, with properties which match the column of the table.
  - The class should use standard Java Bean naming conventions
  - For properties there must be getters and setters
  - Private visibility for the fields (not mandatory)
  - There must be no argument constructor
  - One of the property must be id which holds a unique identifier value (Primary key equivalent)
  - Hibernate can access public,private and protected accessor Methods/ Properties /Default constructor

Two xml files to be configured

- Hibernate Pojo-Table mapping file and the respecive dtd
  - *mypojo.hbm.xml*
  - *hibernate-mapping-3.0.dtd*

- Hibernate Database configuration file and the respective dtd
  - *hibernate.cfg.xml*
  - *hibernate-configuration-3.0.dtd*

mypojo.hbm.xml

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

<!DOCTYPE hibernate-mapping SYSTEM "hibernate-mapping-
    3.0.dtd" >

<hibernate-mapping>

    ----------------------------------

</hibernate-mapping>
```

Hibernate Mapping - EXAMPLE

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping SYSTEM "hibernate-
  mapping-3.0.dtd">
<hibernate-mapping>
<! --mapping of class with table -- >
  <class name="tab1.Tab1" table="tab1">
      <! -- primary key column mapping -- >
   <id name="id" column="id" type="int">
      <generator class="assigned" />
   </id>
      <! -- property mapped with column of table -->
   <property name="name" update="false" insert="true"
      column="name" />
  </class>
</hibernate-mapping>
```

hibernate.cfg.xml

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

<!DOCTYPE hibernate-configuration SYSTEM "hibernate-
    configuration-3.0.dtd" >

<hibernate-configuration>

  <session-factory/>

</hibernate-configuration>
```

Configuration - EXAMPLE

```
<!DOCTYPE hibernate-configuration SYSTEM "src/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
            <property name="connection.driver_class">
                  oracle.jdbc.driver.OracleDriver
            </property>
            <property name="connection.url">
            jdbc:oracle:thin:@192.168.2.4:1521:oracle
            </property>
      <property name="connection.username">
             uname
            </property>
      <property name="connection.password">
             password
      </property>
            <property name="connection.pool_size">
           1
            </property>
```

Configuration xml file contd…

```
    <property name="current_session_context_class">
          thread
    </property>
    <property name="dialect">
                  org.hibernate.dialect.OracleDialect
    </property>
  <!-- org.hibernate.dialect.DB2Dialect  -->
    <property name="show_sql">
    true
    </property>
          <mapping resource="tab1/Tab1.hbm.xml" />
   </session-factory>
</hibernate-configuration>
```

Connecting - SessionFactory

```
SessionFactory sesFactory;
sesFactory =
    new Configuration() . configure() .
                                buildSessionFactory();
Session ses =
sessionFactory . getCurrentSession();
```

Insertion

```
ses.beginTransaction();

Customer cust=new Customer();
cust.setCid(1001);
cust.setCname("raj");
ses.saveOrUpdate(cust);

ses.getTransaction().commit();
```

Selection

```
Query q=ses.createQuery("from Customer");
ses.beginTransaction();
        List result = ses.createQuery("from Customer").list();
            for (int i = 0; i < result.size(); i++)
            {
                Customer cust = (Customer) result.get(i);

                    // Business Logic for handling customer object

            }
             ses.getTransaction().commit();
```

Hibernate Types

A Hibernate Type is used to map a Java property type to a JDBC type or types.

Java Class Attribute Type	*Hibernate Type*
Integer, int, **long short**	integer, long, **short**
char Character	**char**
jBigDecimal	big_decimal
float, **double** float,	**double** float
java.lang.Boolean, **boolean**	boolean
java.lang.string	string
Very **long** strings	text
java.util.Date	date, time, timestamp
java.util.Calendar	calendar

Hibernate Database Dialects

DB2	org.hibernate.dialect.DB2Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
MySQL	org.hibernate.dialect.MySQLDialect
Oracle	org.hibernate.dialect.OracleDialect
Oracle 9i/10g	org.hibernate.dialect.Oracle9Dialect
Sybase	org.hibernate.dialect.SybaseDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect
Ingres	org.hibernate.dialect.IngresDialect

Hibernate JDBC Properties

Property name	Purpose
`hibernate.connection.driver_class`	*jdbc driver class*
`hibernate.connection.url`	*jdbc URL*
`hibernate.connection.username`	*database user*
`hibernate.connection.password`	*database user password*
`hibernate.connection.pool_size`	*maximum number of pooled connections*

Hibernate Datasource Properties

Propery name	Purpose
`hibernate.connection.datasource`	*datasource JNDI name*
`hibernate.jndi.url`	*URL of the JNDI provider* (optional)
`hibernate.jndi.class`	*class of the JNDI* `InitialContextFactory` (optional)
`hibernate.connection.username`	*database user* (optional)
`hibernate.connection.password`	*database user password* (optional)

Hibernate Configuration Properties
Property name and Purpose

hibernate.dialect

- The classname of a Hibernate Dialect which allows Configuration

hibernate.show_sql

- Write all SQL statements to console. This is an alternative to setting the log category

org.hibernate.SQL

- to debug.
- *eg.* true | false

hibernate.format_sql

- Pretty print the SQL in the log and console.
- *eg.* true | false

Hibernate Configuration Properties
Property name and Purpose

hibernate.default_schema

- Qualify unqualified tablenames with the given schema/tablespace in generated SQL.
- *eg.* SCHEMA_NAME

hibernate.default_catalog

- Qualify unqualified tablenames with the given catalog in generated SQL.
- *eg.* CATALOG_NAME

Hibernate Query Language HQL

The Where clause

```
session.createQuery
("from Category c where c.name like 'Laptop%'");
Criteria API
session.createCritera(Category.class)
        .add(Restriction.like("name", "Laptop%"));
```

Uni Directional Mapping
Join
One to Many Relationship

One to Many - Insertion

```
ses.beginTransaction();
    Customer cust=new Customer();
    cust.setCid(1001);
    cust.setCname("mohan");
     HashSet set=new HashSet();
      Product p=new Product();
      p.setCust(cust);
      p.setCid(cust.getCid());
      p.setPid(102);
      p.setPname("pin");
      set.add(p); //one to many example insertion
      cust.setProducts(set);
      ses.saveOrUpdate(cust);
ses.getTransaction().commit();
```

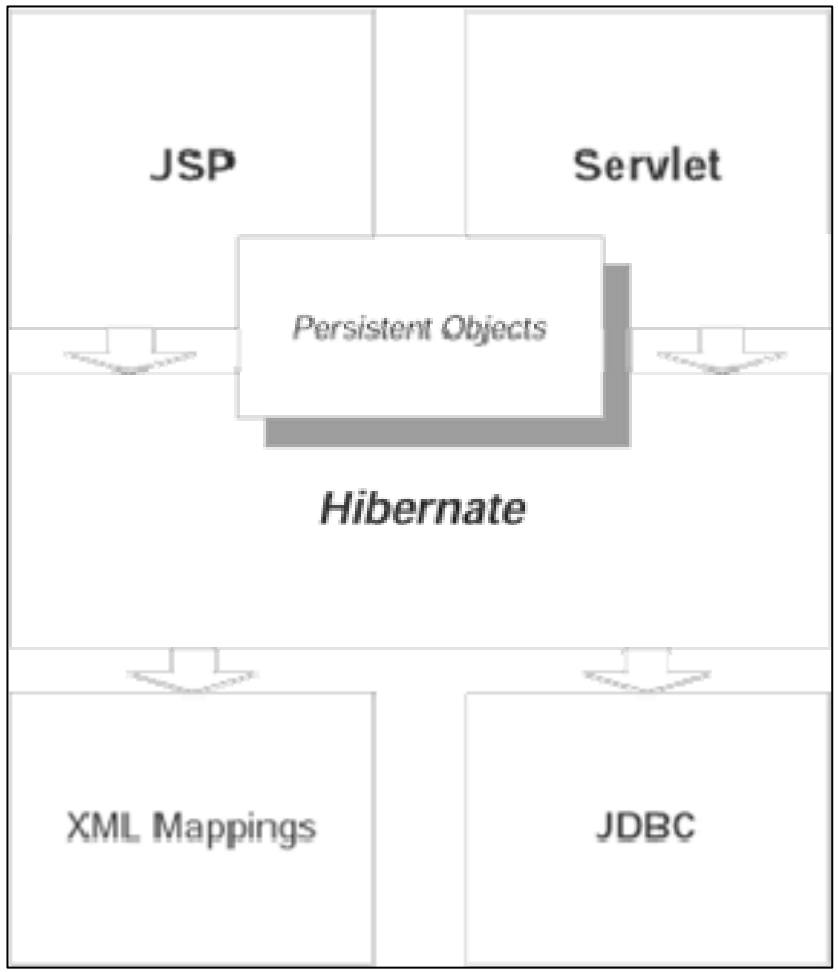
One to Many - Selection

```
ses.beginTransaction();
List result = ses.createQuery("from Customer").list();
Product p=new Product();
  for (int i = 0; i < result.size(); i++) {
            Customer theEvent = (Customer) result.get(i);
           Set set=new HashSet();
           set=theEvent.getProducts();
           Iterator iter=set.iterator();
          while(iter.hasNext()) {
                p=(Product)iter.next();
              System.out.print( theEvent.getCid() +
                   "   " + theEvent.getCname());
            System.out.println("   "+p.getPname());    }   }
        ses.getTransaction().commit();
```

One to Many Mapping in xml

```
<class name="pack1.Customer" table="customer">
    <id name="cid" column="cid" type="int">
          <generator class="assigned"/>
    </id>
    <property name="cname" column="cname"/>
    <set name="products" cascade="all" inverse="true" lazy="true">
       <key column="cid" not-null="true"/>
       <one-to-many class="pack1.Product"/>
   </set>
</class>
<class name="pack1.Product" table="product">
    <id name="pid" column="pid" type="int">
     <generator class="assigned"/>
    </id>
    <property name="pname" column="pname"/>
    <property name="cid" column="cid"/>
</class>
```

Hibernate for Web based Applications



In Summary…

- *ORM tools compared with traditional JDBC tools*
- *Introduction to Hibernate*
- *Hibernate Configurations and Sessions*
- *Mapping a POJO to a table*
- *Association Mapping*
- *Working with data from the Database*
- *Querying data*
- *HQL*
- Using Native SQL

Resources

- Hibernate Home Site (www.hibernate.org)
- Hibernate Tutorial download from hibernate.org

Questions !!!
Thank U!!