H2K Infosys, LLC
Dream, Strive & Achieve Victory

H2K Infosys, LLC
Dream, Strive & Achieve Victory

H2K Infosys, LLC
Dream, Strive & Achieve Victory

www.H2KINFOSYS.com
USA: 770-777-1269
Training@H2KInfosys.com
UK : (020) 3371 7615

HIBERNATE

Part I

Pros and Cons of JDBC

Pros of JDBC	Cons of JDBC
• Clean and simple SQL processing • Good performance with large data • Very good for small applications • Simple syntax so easy to learn	• Complex if it is used in large projects • Large programming overhead • No encapsulation • Hard to implement MVC concept • Query is DBMS specific

Why Object Relational Mapping (ORM)?

When we work with object-oriented systems, there's a mismatch between the object model and the relational database. RDBMSs represent data in a tabular format whereas object-oriented languages, such as Java represent it as an interconnected graph of objects.

In Java:

```
public class Employee {
   private int id;
   private String first_name;
   private String last_name;
   private int salary;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
      this.first_name = fname;
      this.last_name = lname;
      this.salary = salary;
   }
   public int getId() {
      return id;
   }
   public String getFirstName() {
      return first_name;
   }
   public String getLastName() {
      return last_name;
   }
   public int getSalary() {
      return salary;
   }
}
```

In RDBMS:

Emp Id	FirstName	LastName	Salary
110112	Paul	Bulson	66700
112334	Scott	McDerman	78900

ORM stands for Object-Relational Mapping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java

Advantages of ORM:

S.N.	Advantages
1	Let's business code access objects rather than DB tables.
2	Hides details of SQL queries from OO logic.
3	Based on JDBC 'under the hood'
4	No need to deal with the database implementation.
5	Entities based on business concepts rather than database structure.
6	Transaction management and automatic key generation.
7	Fast development of application.

An ORM solution consists of the following four entities:

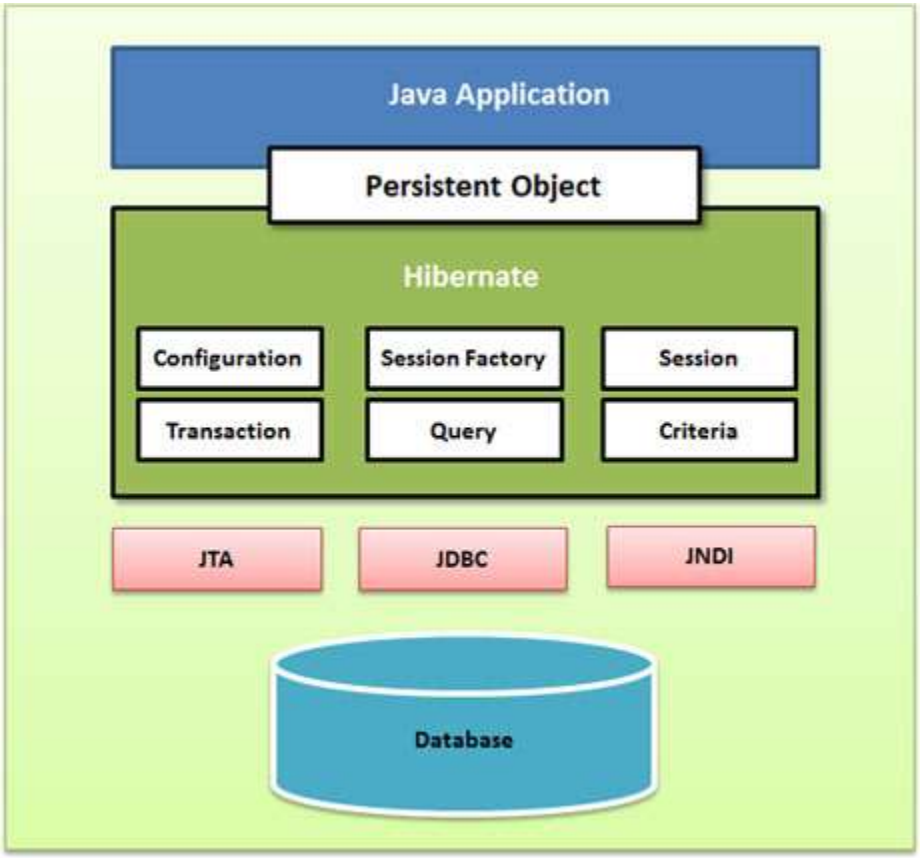
S.N.	Solutions
1	An API to perform basic CRUD operations on objects of persistent classes.
2	A language or API to specify queries that refers to classes and properties of classes.
3	A configurable facility for specifying mapping metadata.
4	A technique to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions.

Hibernate is an Object-Relational Mapping(ORM) solution for JAVA and it raised as 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 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 Database or in any table then the only need to change XML file properties.
- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
- Hibernate does not require an application server to operate.
- Manipulates Complex associations of objects of your database.
- Minimize database access with smart fetching strategies.
- Provides simple querying of data.

The Hibernate architecture is layered to keep you isolated from 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.



Configuration Object:

The Configuration object is the first Hibernate object you create in any Hibernate application and 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.

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 heavyweight object so usually it is created during application start up and kept for later use.

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.

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).

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.

Criteria Object:

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

Hibernate Configuration:

Hibernate requires to know in advance where to find the mapping information that defines how your Java classes relate to the database tables. All such information is usually supplied as standard Java properties file called **hibernate.properties**, or as an XML file named **hibernate.cfg.xml**. This file is kept in the root directory of your application's classpath.

Properties.	Description
hibernate.dialect	This property makes Hibernate generate the appropriate SQL for the chosen database.
hibernate.connection.driver_class	The JDBC driver class.
hibernate.connection.url	The JDBC URL to the database instance.
hibernate.connection.username	The database username.

hibernate.connection.password	The database password.
hibernate.connection.pool_size	Limits the number of connections waiting in the Hibernate database connection pool.
hibernate.connection.autocommit	Allows autocommit mode to be used for the JDBC connection.

If you are using a database along with an application server and JNDI then you would have to configure the following properties:

Properties.	Description
hibernate.connection.datasource	The JNDI name defined in the application server context you are using for the application.
hibernate.jndi.class	The InitialContext class for JNDI.
hibernate.jndi.<JNDIpropertyname>	Passes any JNDI property you like to the JNDI *InitialContext*.
hibernate.jndi.url	Provides the URL for JNDI.
hibernate.connection.username	The database username.
hibernate.connection.password	The database password.

Hibernate with MySQL Database

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
  <property name="hibernate.dialect">
    org.hibernate.dialect.MySQLDialect
  </property>
  <property name="hibernate.connection.driver_class">
    com.mysql.jdbc.Driver
  </property>

  <!-- Assume test is the database name -->
  <property name="hibernate.connection.url">
    jdbc:mysql://localhost/test
  </property>
  <property name="hibernate.connection.username">
    root
  </property>
  <property name="hibernate.connection.password">
    root123
  </property>

  <!-- List of XML mapping files -->
  <mapping resource="Employee.hbm.xml"/>

</session-factory>
```

```
</hibernate-configuration>
```

Dialect Property

Database	Dialect Property
DB2	org.hibernate.dialect.DB2Dialect
HSQLDB	org.hibernate.dialect.HSQLDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Informix	org.hibernate.dialect.InformixDialect
Ingres	org.hibernate.dialect.IngresDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Microsoft SQL Server 2000	org.hibernate.dialect.SQLServerDialect
Microsoft SQL Server 2005	org.hibernate.dialect.SQLServer2005Dialect
Microsoft SQL Server 2008	org.hibernate.dialect.SQLServer2008Dialect
MySQL	org.hibernate.dialect.MySQLDialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle 11g	org.hibernate.dialect.Oracle10gDialect
Oracle 10g	org.hibernate.dialect.Oracle10gDialect
Oracle 9i	org.hibernate.dialect.Oracle9iDialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
Progress	org.hibernate.dialect.ProgressDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect

Hibernate Sessions

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.

```
Session session = factory.openSession();
Transaction tx = null;
try {
   tx = session.beginTransaction();
   // do some work
   ...
   tx.commit();
}
catch (Exception e) {
   if (tx!=null) tx.rollback();
   e.printStackTrace();
}finally {
```

```
    session.close();
}
```

Session Interface Methods

Method	Description
Transaction beginTransaction()	Begin a unit of work and return the associated Transaction object.
void cancelQuery()	Cancel the execution of the current query.
void clear()	Completely clear the session.
Connection close()	End the session by releasing the JDBC connection and cleaning up.
Criteria createCriteria(Class persistentClass)	Create a new Criteria instance, for the given entity class, or a superclass of an entity class
Criteria createCriteria(String entityName)	Create a new Criteria instance, for the given entity name.
Serializable getIdentifier(Object object)	Return the identifier value of the given entity as associated with this session.
Query createFilter(Object collection, String queryString)	Create a new instance of Query for the given collection and filter string
Query createQuery(String queryString)	Create a new instance of Query for the given HQL query string.
SQLQuery createSQLQuery(String queryString)	Create a new instance of SQLQuery for the given SQL query string.
void delete(Object object)	Remove a persistent instance from the datastore.
void delete(String entityName, Object object)	Remove a persistent instance from the datastore.
SessionFactory getSessionFactory()	Get the session factory which created this session
void refresh(Object object)	Re-read the state of the given instance from the underlying database
Serializable save(Object object)	Persist the given transient instance, first assigning a generated identifier
void saveOrUpdate(Object object)	Either save(Object) or update(Object) the given instance.
void update(Object object)	Update the persistent instance with the identifier of the given detached instance.
void update(String entityName, Object object)	Update the persistent instance with the identifier of the given detached instance

Hibernate Persistent Class

Java classes whose objects or instances will be stored in database tables are called persistent classes in Hibernate. Recommended rules for Persistent classesare:

- All Java classes that will be persisted need a default constructor.
- All classes should contain an ID in order to allow easy identification of your objects within Hibernate and the database. This property maps to the primary key column of a database table.

- All attributes that will be persisted should be declared private and have getXXX and setXXX methods defined in the JavaBean style.
- A central feature of Hibernate, proxies, depends upon the persistent class being either non-final, or the implementation of an interface that declares all public methods.

```
public class Employee {
  private int id;
  private String firstName;
  private String lastName;
  private int salary;

  public Employee() {}
  public Employee(String fname, String lname, int salary) {
    this.firstName = fname;
    this.lastName = lname;
    this.salary = salary;
  }
  public int getId() {
    return id;
  }
  public void setId( int id ) {
    this.id = id;
  }
  public String getFirstName() {
    return firstName;
  }
  public void setFirstName( String first_name ) {
    this.firstName = first_name;
  }
  public String getLastName() {
    return lastName;
  }
  public void setLastName( String last_name ) {
    this.lastName = last_name;
  }
  public int getSalary() {
    return salary;
  }
  public void setSalary( int salary ) {
    this.salary = salary;
  }
}
```

Hibernate Mapping Files

An Object/relational mappings are usually defined in an XML document. This mapping file instructs Hibernate how to map the defined class or classes to the database tables.

Considering Program given above and DDL like below:

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

Hibernate mapping should look like below:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

Let's understand few important tags in this XML:

XML Tag	Description
<hibernate-mapping>	Root tag
<class>	mappings from a Java classes to the database tables
<meta>	(optional) create the class description
<id>	Maps the unique ID attribute in class to the primary key of the database table. The **name** attribute of the id element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.
<generator>	Element within the id element is used to automatically generate the primary key values. Set the **class** attribute of the generator element is set to **native** to let hibernate pick up **identity, sequence** or **hilo** algorithm to create primary key depending upon

	the capabilities of the underlying database.
<property>	Used to map a Java class property to a column in the database table. The **name** attribute of the element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type

Hibernate Mapping Types

Primitive Types:

Mapping type	Java type	ANSI SQL Type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT
yes/no	boolean or java.lang.Boolean	CHAR(1) ('Y' or 'N')
true/false	boolean or java.lang.Boolean	CHAR(1) ('T' or 'F')

Date and time types:

Mapping type	Java type	ANSI SQL Type
date	java.util.Date or java.sql.Date	DATE
time	java.util.Date or java.sql.Time	TIME
timestamp	java.util.Date or java.sql.Timestamp	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP
calendar_date	java.util.Calendar	DATE

Let's create a Hibernate Application, which will select, insert, update and delete from Employee table.