Hibernate – I

Basics of Hibernate

Table of Contents

Introduction of Hibernate

1. Hibernate is an open-source ORM(Object/Relational Mapping) solution for Java applications that is developed by Gavin King in 2001.
2. Object/Relational Mapping is a technique of mapping data from an object model representation to a relational data model representation.
3. Hibernate provides data query and retrieval facilities that significantly reduce complexity and development time.
4. Hibernate provides the mapping for association, inheritance, polymorphism, composition, and collections directly with Relational Database.

NOTE: Working with Object-Oriented Software and Relational Databases can be very complex and time consuming. Hibernate makes it very simple, maintainable and reduce development time.

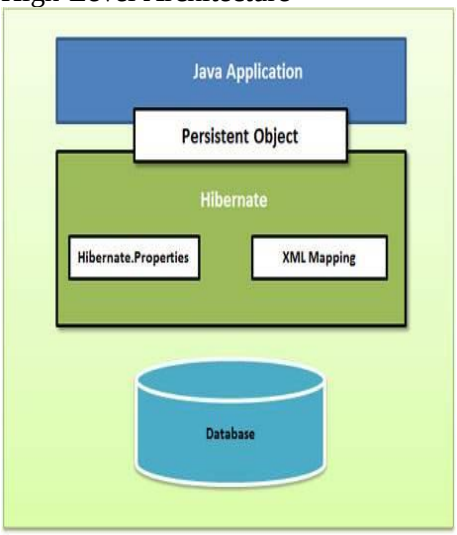
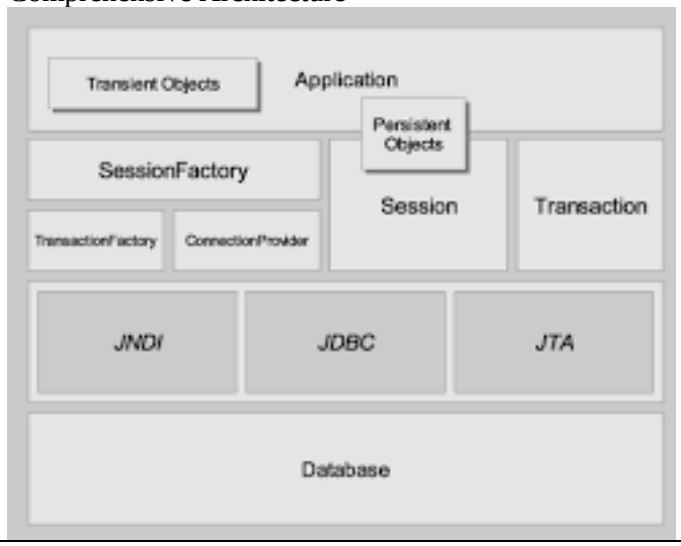
Advantages of Hibernate Over JDBC

1. Database Independence
2. Collection Mapping
3. Association Mapping
4. Inheritance Mapping
5. HQL(Hibernate Query Language)
6. Lazy Loading
7. Auto Update Schema
8. Hibernate Cache
9. Exception Handling
10. Auto Genarated Primary Key

Hibernate Core API's

1. org.hibernate.cfg.Configuration
2. org.hibernate.SessionFactory
3. org.hibernate.Session
4. org.hibernate.Transaction
5. org.hibernate.Query
6. org.hibernate.Criteria

Hibernate Architecture

High-Level Architecture	Comprehensive Architecture
	

Hibernate Environment Setup

JDK does not provide the facilities of the hibernate, so to use hibernate first we need to download the hibernate libraries from the hibernate official site- www.hibernate.org . After downloading, add these libraries with our Java Application then we can use Hibernate Framework.
There are several ways for Hibernate Environment Setup-

1. Using CLASSPATH Environment Variable
2. Using EXT folder of JRE
3. Using Build Path in Eclipse
4. Using Maven Project Management Tool

Persistent Class

Hibernate persistent class uses standard JavaBean naming conventions for property getter and setter methods, as well as private visibility for the fields. This is also called POJO(Plain Old Java Object), Model class and Entity class.

Customer.java

```
public class Customer
{
   private int custId;
   private String custName;
   private String mobile;

   public int getCustId() {
      return custId;
   }
   public void setCustId(int custId) {
      this.custId = custId;
   }
   public String getCustName() {
      return custName;
   }
   public void setCustName(String custName) {
      this.custName = custName;
   }
   public String getMobile() {
      return mobile;
   }
   public void setMobile(String mobile) {
      this.mobile = mobile;
   }
}
```

Hibernate Mapping

Hibernate Mapping file tells to hibernate that how to load and store objects of the persistent class. The mapping file tells Hibernate what table in the database it has to access, and what columns in that table it should use. We save this file with bean_class_name.hbm.xml .

Customer.hbm.xml

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

```
<hibernate-mapping package="com.kalibermind.hibernate.bean">
    <class name="Customer" table="CUSTOMER">
    <id name="custId">
       <generator class="identity" />
    </id>
    <property name="custName" />
    <property name="mobile" />
  </class>
</hibernate-mapping>
```

Hibernate DTD is not optional you must have to define. It provides auto-completion of XML mapping elements and attributes in your editor or IDE.

Hibernate will not load the DTD file from the web, but first look it up from the classpath of the application. The DTD file is included in hibernate-core.jar

Hibernate Configuration

Hibernate configuration is defined by hibernate.cfg.xml file. This is the main configuration file of hibernate framework that consists datasource information, hibernate properties and mapping resources. The default location of this file is CLASSPATH .

hibernate.cfg.xml

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

    <!-- Database connection settings -->
    <property name="driver_class">com.mysql.jdbc.Driver</property>
    <property name="url">jdbc:mysql://localhost:3306/hibernatedb</property>
    <property name="username">root</property>
    <property name="password">root</property>

    <!-- SQL dialect -->
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">true</property>

    <!-- Create the database schema on startup -->
    <property name="hbm2ddl.auto">update</property>

    <mapping resource="Customer.hbm.xml"/>
  </session-factory>

</hibernate-configuration>
```

Hibernate Dialects

RDBMS	Dialect
DB2	org.hibernate.dialect.DB2Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
MySQL	org.hibernate.dialect.MySQLDialect
MySQL with InnoDB	org.hibernate.dialect.MySQLInnoDBDialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle 9i	org.hibernate.dialect.Oracle9iDialect
Oracle 10g	org.hibernate.dialect.Oracle10gDialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect

Hibernate Generator Classes

A generator class defines the primary key of the generated table. All the generator classes are belong to **org.hibernate.id.*;** package and implement the interface **org.hibernate.id.IdentifierGenerator.** Here we Are discussing frequently used generator classes.

Generator Class	Description
increment	Generates identifiers of type **long,int or short**
assigned	Application assign an identifier to the object before save().
identity	Supported by MySQL,DB2,MS SQL Server,Sybase and HSQL
sequence	Supported by Oracle, PostgreSQL, DB2, SAP DB, McKoi
native	Selects identity, sequence or hilo depending upon Database.
hilo	Uses a hi/lo algorithm to generate identifiers of type long,int or short.
uuid	Generates a 128-bit UUID based on a custom algorithm.

First Application Using XML

Required Component to develop a Hibernate application

1. POJO class **(Plain old java object)**
2. Mapping file **(Customer.hbm.xml)**
3. Configuration file **(.cfg.xml)**
4. Test class is required

Customer.java

```
package com.Biditvats.domain;

public class Customer {
  private Long id;
  private String firstName;
  private String lastName;
```

```
  private String email;
  private Long mobile;

  public Customer() {
        System.out.println("Customer object is created");
  }

 public Customer(String firstName, String lastName, String email, Long mobile) {

      this.firstName = firstName;
      this.lastName = lastName;
      this.email = email;
      this.mobile = mobile;
 }
       //corresponding getters
      //corresponding setters
 }
```

Customer.hbm.xml

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

<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
      <class name="com.Biditvats.domain.Customer" table="CUSTOMER_MASTER1">
            <!-- Primary Key -->
            <id name="id" column="CUSTOMER_ID">
                  <generator class="identity"></generator>
            </id>

            <property name="firstName" column="FIRST_NAME" />
            <property name="lastName" column="LAST_NAME" />
            <property name="email" column="EMAIL" />
            <property name="mobile" column="MOBILE" />

      </class>
</hibernate-mapping>
```

hibernate.cfg.xml

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

<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

      <session-factory>
            <!-- Data Source Details -->
            <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
            <propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatedb</property>
            <property name="hibernate.connection.username">root</property>
```

```
            <property name="hibernate.connection.password">password</property>

            <!-- Hibernate Properties -->
            <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
            <property name="hibernate.hbm2ddl.auto">update</property>
            <property name="hibernate.show_sql">true</property>

            <!-- Resource Mapping -->
            <mapping resource="Customer.hbm.xml" />
        </session-factory>

    </hibernate-configuration>
```

SaveCustomer.java

```
    package com.Biditvats.test;

    import org.hibernate.Session;
    import org.hibernate.SessionFactory;
    import org.hibernate.Transaction;
    import org.hibernate.cfg.Configuration;

    import com.Biditvats.domain.Customer;

    public class SaveCustomer {

        public static void main(String[] args) {
            Configuration configuration = new Configuration();
            configuration.configure();

            SessionFactory factory = configuration.buildSessionFactory();

            Customer customer = new Customer("Bidit","vats","Biditvats@gmail.com",9916712669L);

            Session session = factory.openSession();
            Transaction transaction = session.beginTransaction();
            session.save(customer);
            transaction.commit();
            session.close();

            System.out.println("Record Have been saved successfully!");
        }
    }
```

Hibernate Annotations

Annotation	Description
@Entity	It declares the class as an entity bean (a persistent POJO class).
@Table	It declares the table name (default to the class name)
@Id	It declares the identifier property of the entity bean.
@GeneratedValue	It declares the generator type for identifier property. **@GeneratedValue(strategy=GenerationType.IDENTITY) GenerationType= IDENTITY, SEQUENCE,TABLE,AUTO**
@Column	**name (optional):** the column name (default to the property name)

First Application Using Annotation

Customer.java

```
package com.Biditvats.domain;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="CUSTOMER_TEMP")
public class Customer {

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  @Column(name="CUSTOMER_ID")
  private Long id;

  @Column(name="FIRST_NAME")
  private String firstName;

  @Column(name="LAST_NAME")
  private String lastName;

  @Column(name="EMAIL")
  private String email;

  @Column(name="MOBILE")
  private Long mobile;

  public Customer() {
        System.out.println("Customer object is created");
  }

public Customer(String firstName, String lastName, String email, Long mobile) {

        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.mobile = mobile;
}
//corresponding getters
        //corresponding setters
}
```

hibernate.cfg.xml

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

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
```

```
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
        <!-- Data Source Details -->
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatedb</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">password</property>

        <!-- Hibernate Properties -->
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property name="hibernate.show_sql">true</property>

        <!-- Resource Mapping -->
        <mapping class="com.Biditvats.domain.Customer" />
    </session-factory>

</hibernate-configuration>
```

SaveCustomer.java

```
package com.Biditvats.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.Biditvats.domain.Customer;

public class SaveCustomer {

    public static void main(String[] args) {
        Configuration configuration = new Configuration();
        configuration.configure("hibernate.cfg.xml");

        SessionFactory factory = configuration.buildSessionFactory();

        Customer customer = new Customer("Bidit","vats","Biditvats@gmail.com",9916712669L);

        Session session = factory.openSession();
        Transaction transaction = session.beginTransaction();
        session.save(customer);
        transaction.commit();
        session.close();

        System.out.println("Record Have been saved successfully!");
    }
}
```

Hibernate Object State

Hibernate objects are categorized in **three** state.

Transient Object

- ✓ An object is transient if it has just been instantiated using the new operator, and it is not associated with a Hibernate Session.
- ✓ It has no persistent representation in the database and no identifier value has been assigned.
- ✓ Transient instances will be destroyed by the garbage collector if the application does not hold a reference anymore.

Persistent Object

- ✓ A persistent instance has a representation in the database and an identifier value.
- ✓ It might just have been saved or loaded, however, it is by definition in the scope of a Session.
- ✓ Hibernate will detect any changes made to an object in the persistent state and synchronize the state with the database when the unit of work completes.
- ✓ Developers do not execute manual UPDATE statements, or DELETE statements when an object should be made transient.

Detached Object

- ✓ A detached instance is an object that has been persistent, but its session has been closed.
- ✓ The reference to the object is still valid, of course, and the detached instance might even be modified in this state.
- ✓ A detached instance can be reattached to a new Session at a later point in time, making it persistent again.