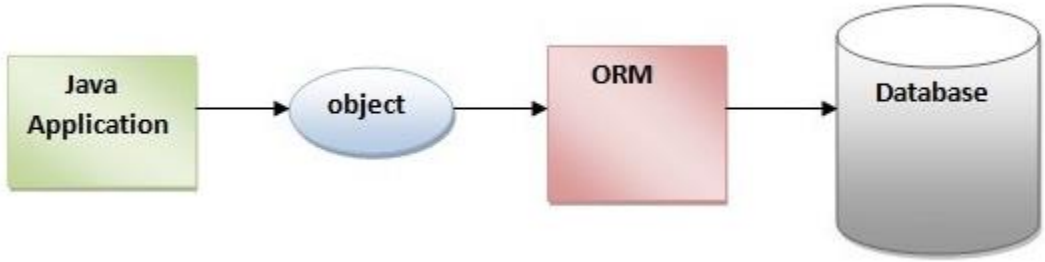
Hibernate Framework

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool.

ORM Tool

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.



Advantages of Hibernate Framework

1) Open Source and Lightweight

Hibernate framework is open source under the LGPL license and lightweight.

2) Fast Performance

The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

3) Database Independent Query

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

4) Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

5) Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

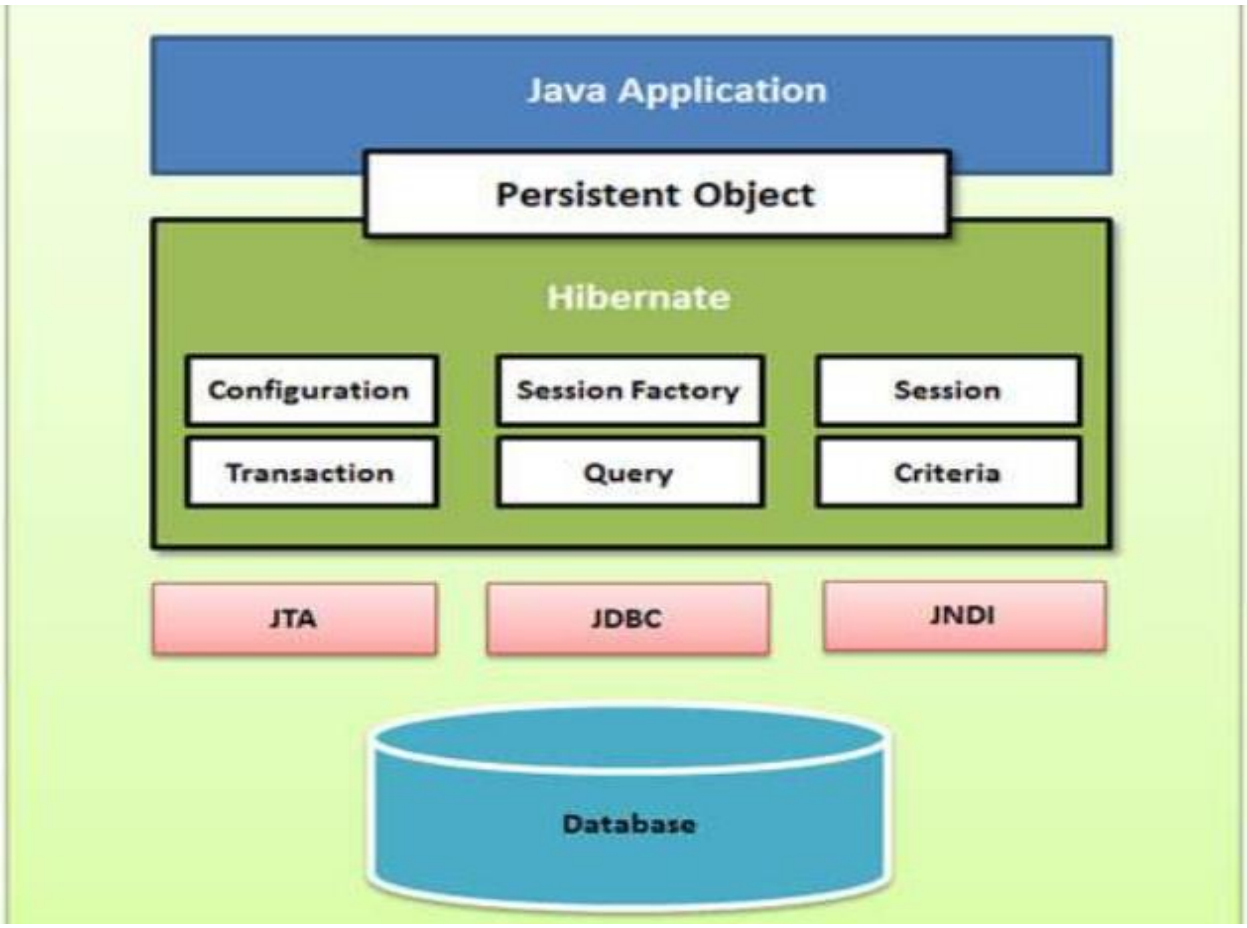
Hibernate Architecture

The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

The Hibernate architecture is categorized in four layers.

- Java application layer
- Hibernate framework layer
- Backhand api layer
- Database layer

Elements of Hibernate Architecture



For creating the first hibernate application, we must know the elements of Hibernate architecture.

SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data.

Session

A Session is used to get a physical connection with a database. The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data.

Transaction

The transaction object specifies the atomic unit of work. It is optional. This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code. The org.hibernate.Transaction interface provides methods for transaction management.

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.

ConnectionProvider

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource.

Hibernate First Level Cache

Hibernate Cache can be very useful in gaining fast application performance if used correctly. The idea behind cache is to reduce the number of database queries, hence reducing the throughput time of the application.
Hibernate first level cache is associated with the Session object. Hibernate first level cache is enabled by default and there is no way to disable it. However hibernate provides methods through which we can delete selected objects from the cache or clear the cache completely. Any object cached in a session will not be visible to other sessions and when the session is closed, all the cached objects will also be lost.

TransactionFactory

It is a factory of Transaction. It is optional.

Hibernate Properties

Following is the list of important properties, you will be required to configure for a databases in a standalone situation −

Sr.No.	Properties & Description
1	**hibernate.dialect** This property makes Hibernate generate the appropriate SQL for the chosen database.
2	**hibernate.connection.driver_class** The JDBC driver class.
3	**hibernate.connection.url** The JDBC URL to the database instance.
4	**hibernate.connection.username** The database username.
5	**hibernate.connection.password** The database password.
6	**hibernate.connection.pool_size** Limits the number of connections waiting in the Hibernate database connection pool.
7	**hibernate.connection.autocommit** Allows autocommit mode to be used for the JDBC connection.

Hibernate Example

1. Create the Persistent class
2. Create the mapping file for Persistent class
3. Create the Configuration file
4. Create the class that retrieves or stores the persistent object
5. Load the jar file
6. Run the first hibernate application by using command prompt

Depdendency –

```
<dependency>
     <groupid>org.hibernate</groupid>
     <artifactid>hibernate-annotations</artifactid>
     <version>3.3.0.ga</version>
</dependency>
```

Persistent class -

```
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;
    }

}
```

Mapping Configuration File -

```
<?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>
```

hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-5.3.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">jtp</property>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<mapping resource="employee.hbm.xml"/>
</session-factory>

</hibernate-configuration>
```

Application class –

```
public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {

      try {
         factory = new Configuration().configure().buildSessionFactory();
      } catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object." + ex);
         throw new ExceptionInInitializerError(ex);
      }

      ManageEmployee ME = new ManageEmployee();

      /* Add few employee records in database */
      Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);
      Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
      Integer empID3 = ME.addEmployee("John", "Paul", 10000);

      /* List down all the employees */
      ME.listEmployees();

      /* Update employee's records */
      ME.updateEmployee(empID1, 5000);

      /* Delete an employee from the database */
      ME.deleteEmployee(empID2);

      /* List down new list of the employees */
      ME.listEmployees();
   }

   /* Method to CREATE an employee in the database */
   public Integer addEmployee(String fname, String lname, int salary){
      Session session = factory.openSession();
      Transaction tx = null;
      Integer employeeID = null;
```

```
    try {
        tx = session.beginTransaction();
        Employee employee = new Employee(fname, lname, salary);
        employeeID = (Integer) session.save(employee);
        tx.commit();
    } catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
    } finally {
        session.close();
    }
    return employeeID;
}

/* Method to  READ all the employees */
public void listEmployees( ){
    Session session = factory.openSession();
    Transaction tx = null;

    try {
        tx = session.beginTransaction();
        List employees = session.createQuery("FROM Employee").list();
        for (Iterator iterator = employees.iterator(); iterator.hasNext();){
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName());
            System.out.print("  Last Name: " + employee.getLastName());
            System.out.println("  Salary: " + employee.getSalary());
        }
        tx.commit();
    } catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
    } finally {
        session.close();
    }
}

/* Method to UPDATE salary for an employee */
public void updateEmployee(Integer EmployeeID, int salary ){
    Session session = factory.openSession();
    Transaction tx = null;

    try {
        tx = session.beginTransaction();
        Employee employee = (Employee)session.get(Employee.class, EmployeeID);
        employee.setSalary( salary );
```

```
            session.update(employee);
            tx.commit();
        } catch (HibernateException e) {
            if (tx!=null) tx.rollback();
            e.printStackTrace();
        } finally {
            session.close();
        }
    }

    /* Method to DELETE an employee from the records */
    public void deleteEmployee(Integer EmployeeID){
        Session session = factory.openSession();
        Transaction tx = null;

        try {
            tx = session.beginTransaction();
            Employee employee = (Employee)session.get(Employee.class, EmployeeID);
            session.delete(employee);
            tx.commit();
        } catch (HibernateException e) {
            if (tx!=null) tx.rollback();
            e.printStackTrace();
        } finally {
            session.close();
        }
    }
}
```