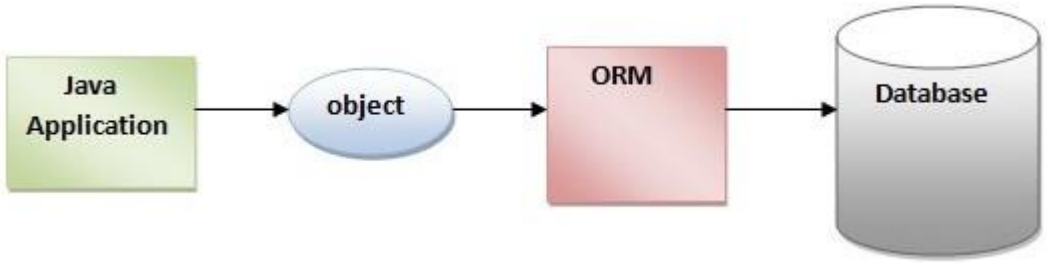
Hibernate Tutorial

- Hibernate is an open source Java persistence framework project.
- It performs powerful object-relational mapping and query databases using HQL and SQL.
- Hibernate is a great tool for ORM mappings in Java.
- It can cut down a lot of complexity and thus defects as well from your application, which may otherwise find a way to exist.
- This is specially boon for developers with limited knowledge of SQL.
- Initially started as an ORM framework, Hibernate has spun off into many projects, such as Hibernate Search, Hibernate Validator, Hibernate OGM (for NoSQL databases), and so on.

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.



The ORM tool internally uses the JDBC API to interact with the database.

What is JPA?

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

Advantages of Hibernate Framework

Following are the 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.

6) Provides Query Statistics and Database Status

Hibernate supports Query cache and provide statistics about query and database status.

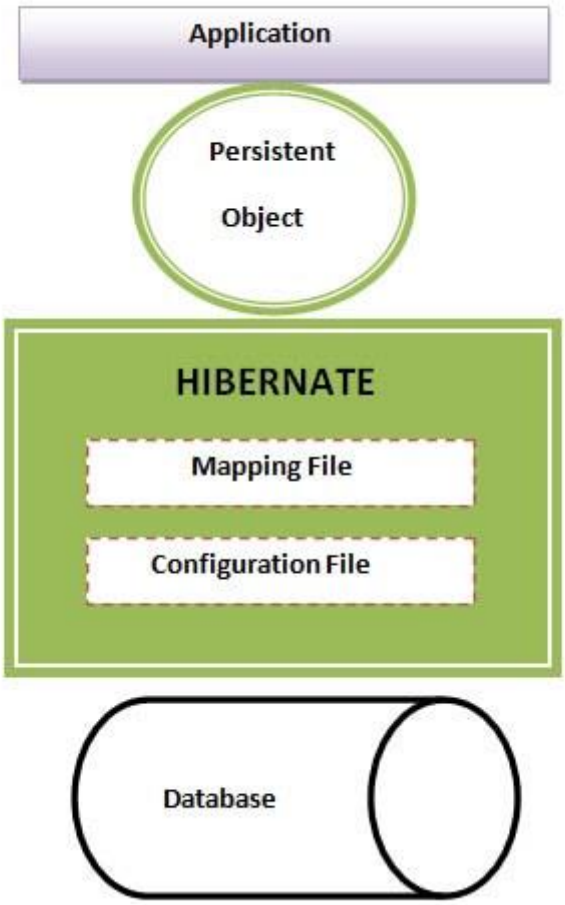
Hibernate Architecture

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

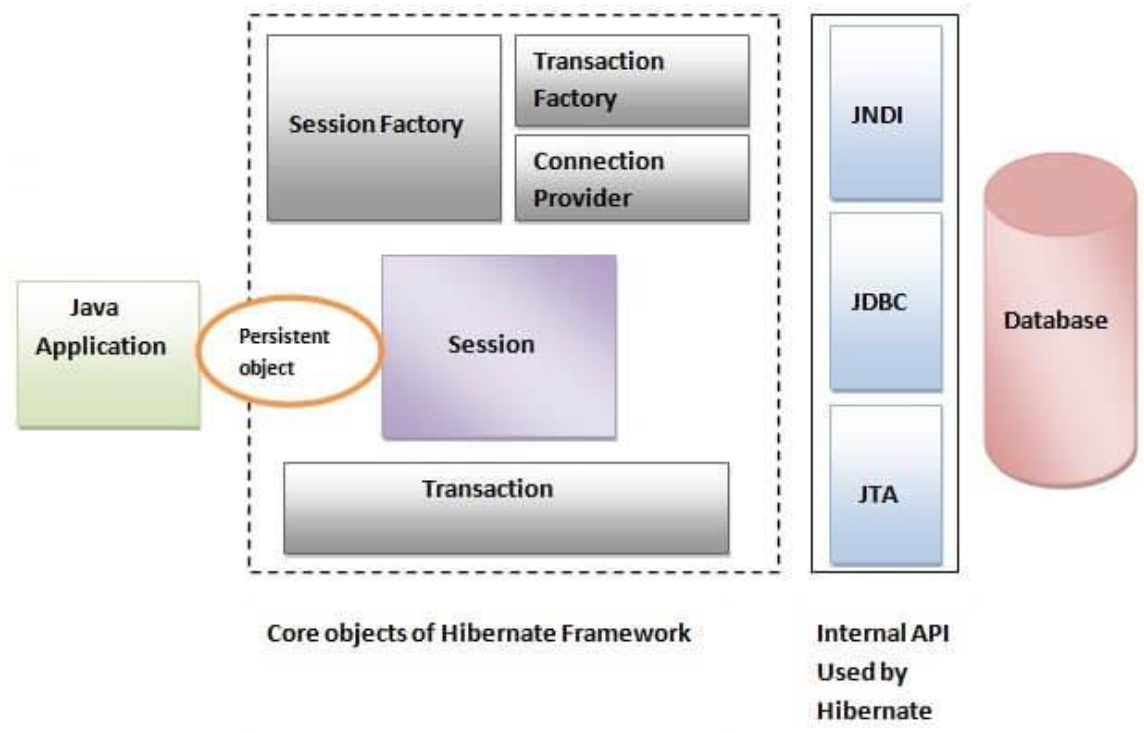
The Hibernate architecture is categorized in four layers.

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

Let's see the diagram of hibernate architecture:



This is the high level architecture of Hibernate with mapping file and configuration file.



Hibernate framework uses many objects such as session factory, session, transaction etc. along with existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

Elements of Hibernate Architecture

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

SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

Session

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. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

Transaction

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

ConnectionProvider

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

TransactionFactory

It is a factory of Transaction. It is optional.

https://www.javatpoint.com/example-to-create-hibernate-application-in-eclipse-ide

First Example :

----hibernate.cfg.xml ---

```
<?xml version="1.0"?>
<!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="connection.driver_class">org.h2.Driver</property>
        <property name="connection.url">jdbc:h2:~/hcldb</property>
        <property name="connection.username">sa</property>
        <property name="connection.password"/>
        <property name="dialect">org.hibernate.dialect.H2Dialect</property>
        <!--  Echo all executed SQL to stdout  -->
        <property name="show_sql">true</property>
        <!--  Drop and re-create the database schema on startup  -->
        <property name="hbm2ddl.auto">create-drop</property>
        <mapping class="com.hcl.model.Message"/>
    </session-factory>

</hibernate-configuration>
```

Master POM

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
```

```
  <groupId>com.hcl.trainings</groupId>
  <artifactId>hibernate-trainings-parent</artifactId>
  <packaging>pom</packaging>
  <version>1.0.0</version>
  <name>hibernate-parent</name>
  <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
      <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>6.14.3</version>
        <scope>test</scope>
      </dependency>
      <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.2.17.Final</version>
      </dependency>
      <dependency>
    <groupId>org.javassist</groupId>
    <artifactId>javassist</artifactId>
    <version>3.25.0-GA</version>
</dependency>
      <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.197</version>
        </dependency>

      <!-- API, java.xml.bind module -->
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
    <version>2.3.2</version>
</dependency>

<!-- Runtime, com.sun.xml.bind module -->
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
    <version>2.3.2</version>
</dependency>
    </dependencies>
    <build>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>2.3.2</version>
          <configuration>
            <source>1.8</source>
            <target>1.8</target>
            <showDeprecation>true</showDeprecation>
            <showWarnings>true</showWarnings>
          </configuration>
```

```
            </plugin>
        </plugins>
    </build>
    <modules>
            <module>day-1</module>
    </modules>
</project>
```

```
package com.hcl.model;

import javax.persistence.*;

@Entity
public class Message {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    Long id;
    @Column(nullable = false)
    String text;

    public Message(String text) {
        setText(text);
    }

    public Message() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    @Override
    public String toString() {
        return "Message{" +
                "id=" + getId() +
                ", text='" + getText() + '\'' +
                '}';
    }
}
```

```
package com.hcl.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

import com.hcl.model.Message;

import java.util.List;

import static org.testng.Assert.assertEquals;


public class PersistenceTest {
   private SessionFactory factory = null;

   @BeforeClass
   public void setup() {
      StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
            .configure()
            .build();
      factory = new MetadataSources(registry).buildMetadata().buildSessionFactory();
   }

   @Test
   public void saveMessage() {
      Message message = new Message("Hello, world");
      try (Session session = factory.openSession()) {
         Transaction tx = session.beginTransaction();
         session.persist(message);
         tx.commit();
      }
   }

   @Test(dependsOnMethods = "saveMessage")
   public void readMessage() {
      try (Session session = factory.openSession()) {
         List<Message> list = session.createQuery("from Message", Message.class).list();

         assertEquals(list.size(), 1);
         for (Message m : list) {
            System.out.println("!!!!"+m);
         }
      }
   }
}
```