Hibernate Introduction

Lesson Objectives

1. Understand what is Hibernate and when to used it?
2. Describe the architecture and components of the Hibernate framework
3. Hibernate's benefits over using a JDBC connection
4. Use properties in the Hibernate Configuration file
5. Use Session, SessionFactory and Transaction

Agenda

- Hibernate Overview
- Hibernate Features
- Hibernate Architecture
- Configuration
- Hibernate First Example

Section 01

HIBERNATE OVERVIEW

What is ORM?

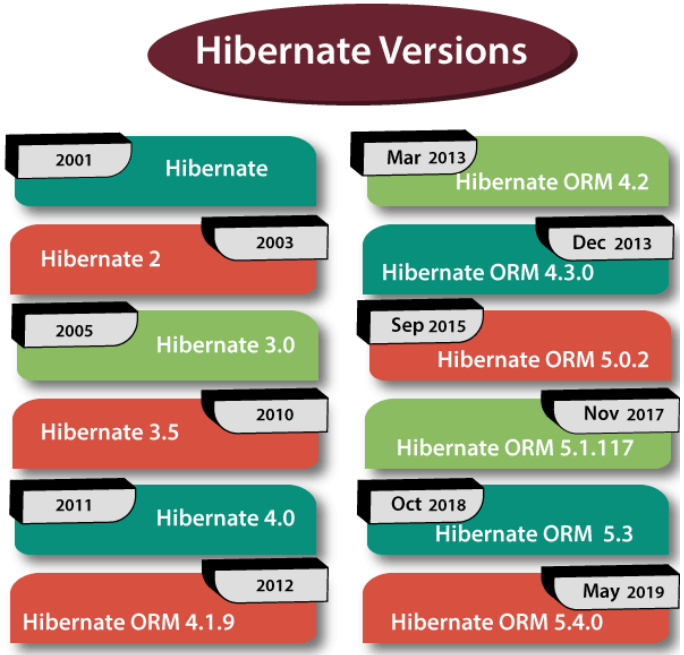
- ORM stands for **O**bject-**R**elational **M**apping (ORM), is a programming technique for **converting data** between **relational databases** and **object oriented programming languages.**
- Java ORM Frameworks:
  - Hibernate
  - Spring DAO
  - Open JPA
  - Mybatis/iBatis
  - Toplink

What is Hibernate?

- Hibernate is an open source persistent framework, lightweight, ORM (**Object Relational Mapping)** tool solution for JAVA.
- Hibernate is a Java framework that simplifies the development of Java application to **interact with the database**.
- Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

History of Hibernate

- Hibernate was developed in 2001 by **Gavin king** with his colleagues from Circus Technologies.

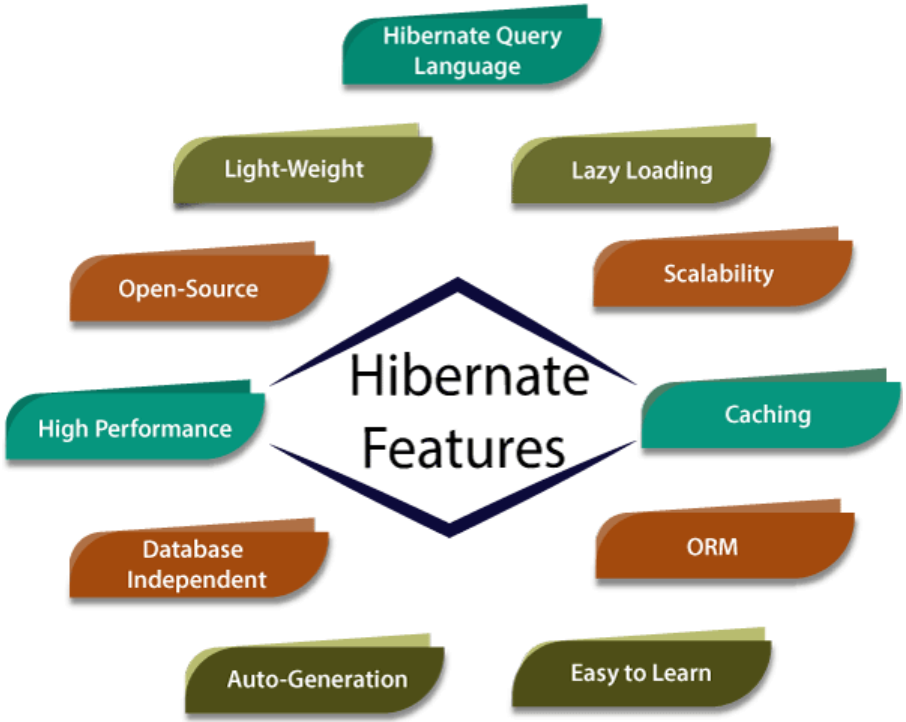


https://www.tutorialandexample.com/

Section 02

HIBERNATE FEATURES

Features of Hibernate



Features of Hibernate

1. Lightweight

- ✓ Hibernate is a lightweight framework as it does not contains additional functionalities; it uses only those functionalities required for object-relational mapping.
- ✓ It is a lightweight framework because it uses persistent classes for data transfer between java application and databases.

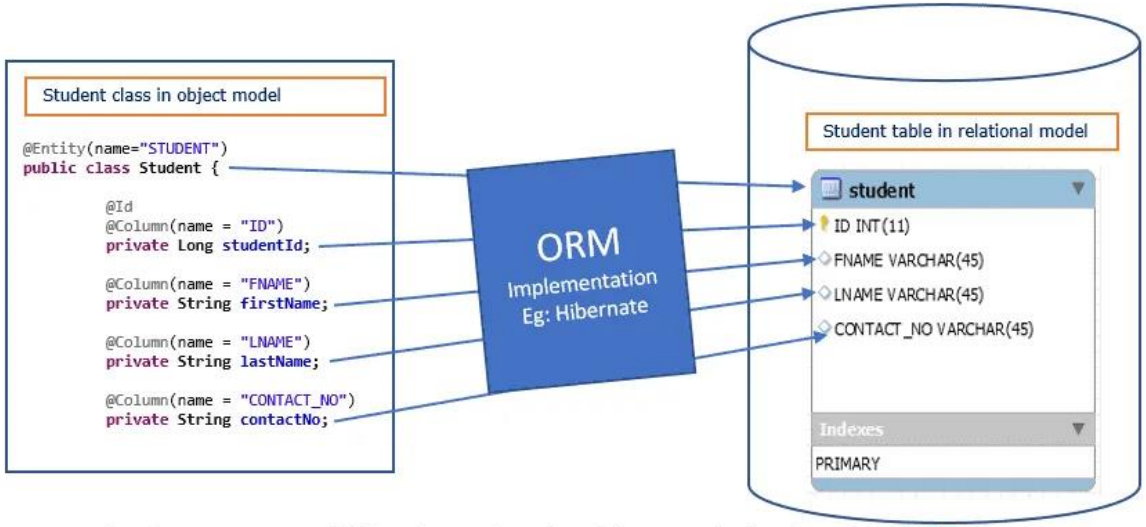
2. Open Source

- ✓ Hibernate is open-source software that means it is available for everyone without any cost.
- ✓ It can be downloaded from its official website, http://hibernate.org/ . The latest version a user can download is Hibernate 5.4.

Features of Hibernate

3. ORM (Object Relation Mapping)

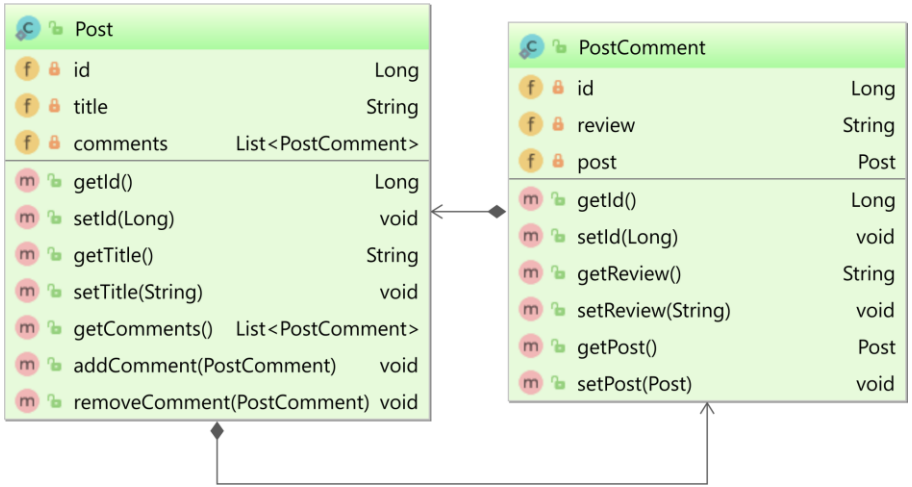
- ✓ Hibernate is an ORM tool which helps in the interaction between the java classes and relational database



Features of Hibernate

4. High Performance

- ✓ Hibernate supports many different fetching techniques such as, caching, lazy initialization, and many more to achieve high performance.



Can not load comment (associated data)

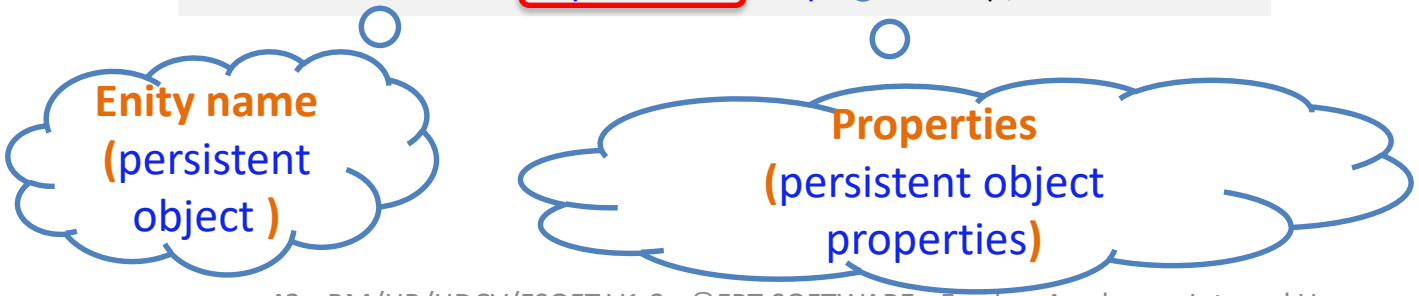
(Associated data loads only when we explicitly call getter or size method.)

Features of Hibernate

5. HQL (Hibernate Query Language)

- ✓ Hibernate has its query language, i.e., HQL (Hibernate Query Language) which is independent of the database.
- ✓ HQL is an object-oriented language similar to SQL, but it works with the persistent object and its properties.

```
Query<User> query = session.createQuery(
        "FROM User u WHERE u.userName = :userName "
        + "AND u.password = :password");
```



Features of Hibernate

6. **Caching**
   - ✓ Hibernate supports **two levels of caching**, first-level and second-level caching.
   - ✓ Caching is the process of storing data into cache memory and improves the speed of data access.
7. **Auto-Generation**
   - ✓ Hibernate provides a feature of automatic table generation.
8. **Scalability**
   - ✓ Hibernate is highly scalable as it can be fit in any environment. Hibernate can be used for both small scale and large scale applications.

Features of Hibernate

9. Lazy Loading

- ✓ Hibernate supports a new concept called lazy loading. Lazy loading concept retrieves the only necessary object for execution.

10. Easy to learn

- ✓ Hibernate is easy to learn and implement.

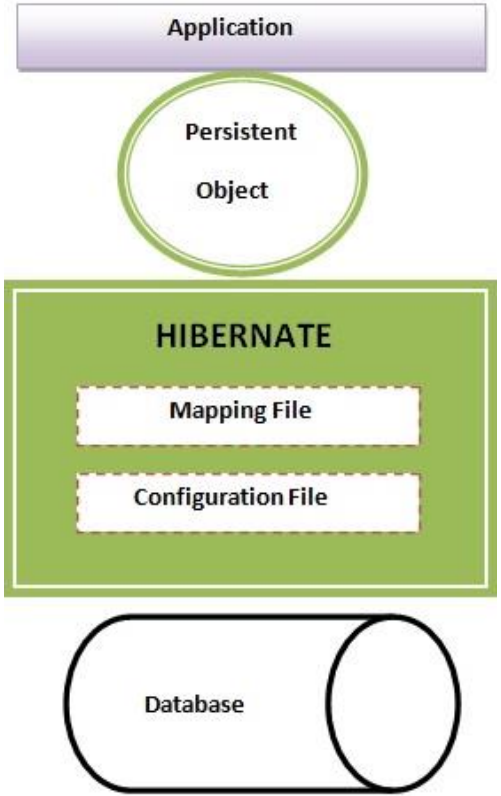
11. Database Independent

- ✓ Hibernate is database-independent as it provides 'Database Dialect' so we need not write SQL queries.
- ✓ It supports many databases such as **Oracle**, **MySql**, **Sybase**, **SQL Server**, etc.

Section 03

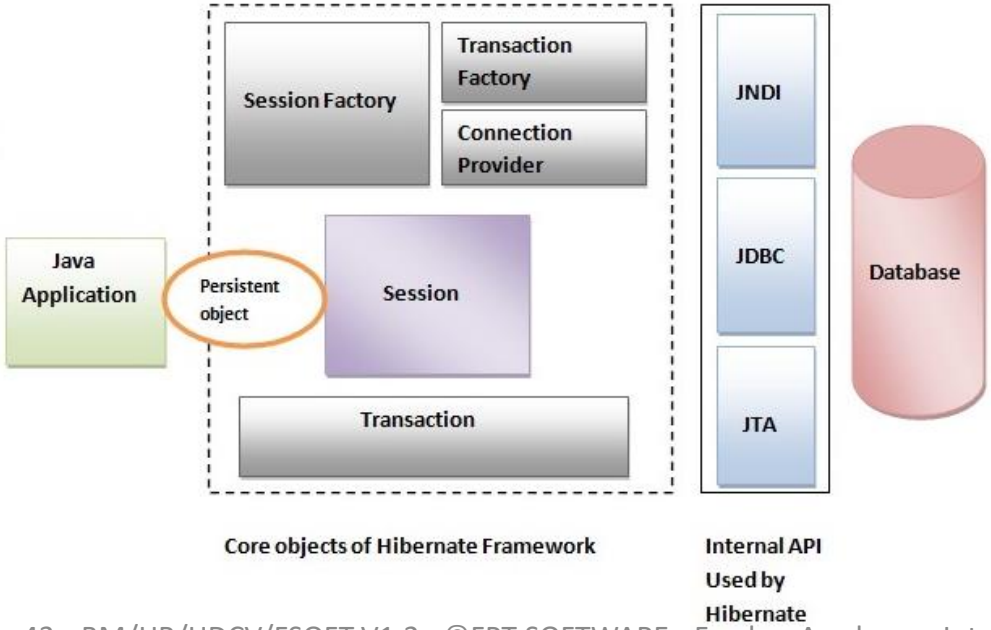
HIBERNATE ARCHITECTURE

High Level Architecture

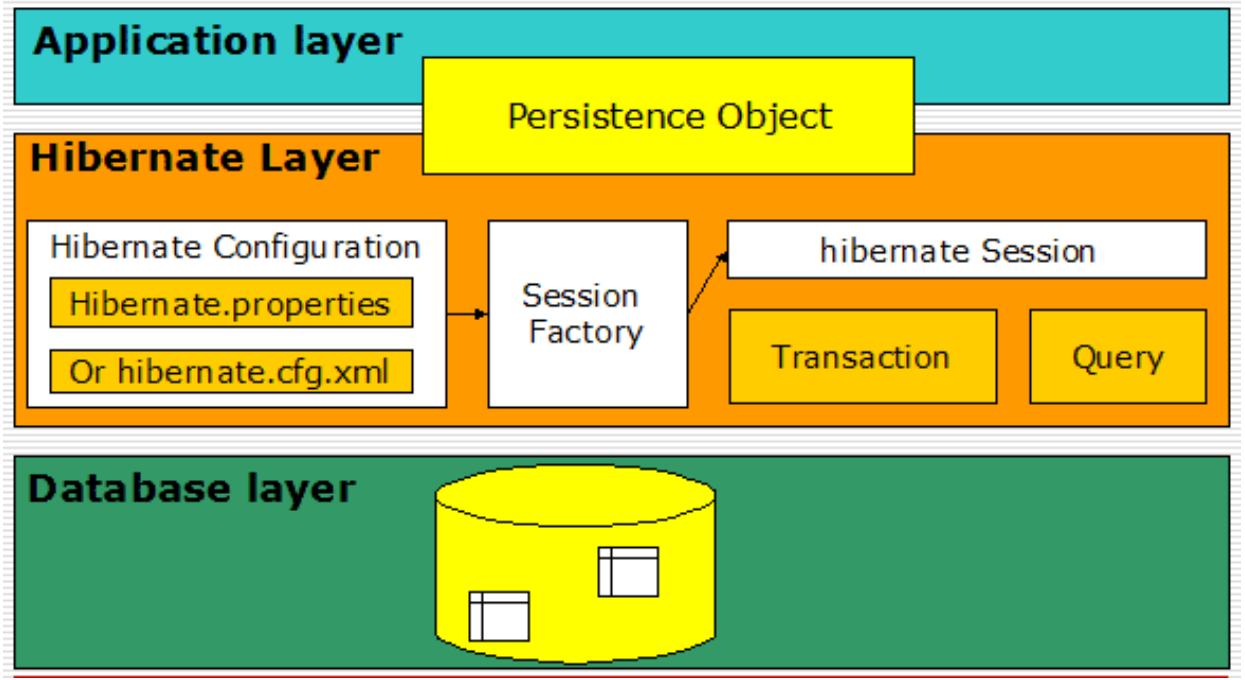


Core Components of Hibernate

- Hibernate framework uses many objects **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).



Core Components of Hibernate



Core Components of Hibernate

- **Configuration**
  - ✓ In package: org. cfg
  - ✓ The Configuration class consists of the properties and function files of
  - ✓ It reads both mapping and configuration file.
  - ✓ **Syntax:**

```
Configuration cfg = new Configuration();
cfg.configure();
```

hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>
        <property name="connection.url">jdbc:sqlserver://localhost:1433;databaseName=hrms</property>
        <property name="connection.username">sa</property>
        <property name="connection.password">12345678</property>

        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

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

    </session-factory>
</hibernate-configuration>
```

Core Components of Hibernate

❖ SessionFactory

- ✓ The **org. sessionFactory** package contains the **SessionFactory** interface whose object can be obtained by the object of **Configuration** class.
- ✓ It is a threadsafe object and used by all the threads in the application.
- ✓ **Syntax:**

```
SessionFactory sessionFactory = cfg.buildSessionFactory();
```

Or

```
ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder().
                                  applySettings(cfg.getProperties()).build();

SessionFactory sessionFactory = cfg.buildSessionFactory(serviceRegistry);
```

- ✓ *It takes the JDBC information from cfg object and creates a JDBC connection.* It provides factory method to **get the object of Session**.

Core Components of Hibernate

❖ SessionFactory methods:

SessionFactory provides **three methods** through which we can get Session object:

- ✓ **openSession()**: method always **opens a new session**. We should close this session object once we are done with all the database operations.
- ✓ **getCurrentSession()**: method returns the session **bound to the context**. Need to have configured in hibernate configuration file like following:

```
<property name="current_session_context_class">thread</property>
```

It's faster than opening a new session

SessionFactory class

❖ SessionFactory methods:

- ✓ openStatelessSession(): method returns instance of **StatelessSession**.
- ✓ StatelessSession in Hibernate does **not implement first-level cache** and it **doesn't interact with any second-level cache**.
- ✓ **Collections are also ignored** by a stateless session.
- ✓ It's more like a normal JDBC connection and doesn't provide any benefits that come from using hibernate framework.

Core Components of Hibernate

❖ Session

- ✓ The session object provides an interface between the ***application*** and ***data stored*** in the database. A Session is used to get a **physical connection with a 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.
- ✓ It is factory of Transaction, Query and Criteria.
- ✓ It holds a first-level cache (mandatory) of data.
- ✓ The ***org. Session*** object is not threadsafe. It is used to execute CRUD operations (insert, delete, update, edit).
- ✓ **Syntax**:

```
Session session = sessionFactory.openSession();
```

Core Components of Hibernate

❖ Transaction

- ✓ The **org.hibernate** package contains a **Transaction** interface.
- ✓ The object of the session creates a Transaction object.
- ✓ It provides the instruction to the database for transaction management.
- ✓ It is a short-lived single-threaded object.

```
Transaction transaction = session.beginTransaction();
Serializable result = session.save(job);
transaction.commit();
```

Section 04

CONFIGURATION

Introduction

- How your **Java classes** relate to the **database tables**?
- Hibernate requires a **set of configuration settings related to database** and other related **parameters**.
- All such information is usually supplied as a standard Java properties file called **properties**, or as an XML file named **hibernate.cfg.xml**.
- We will consider XML formatted file **hibernate.cfg.xml** to specify required Hibernate properties in all of examples.

hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE      hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
               "http:// sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
   <!-- Database connection settings -->
   <property name="connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>
   <property name="connection.url">jdbc:sqlserver://localhost:1433;databaseName=hrms</property>
   <property name="connection.username">sa</property>
   <property name="connection.password">12345678</property>

   <!-- JDBC connection pool (use the built-in) -->
   <property name="connection.pool_size">1</property>

   <!-- SQL dialect -->
   <property name="dialect">org. dialect.SQLServerDialect</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">update</property>
</session-factory>
</hibernate-configuration>
```

Database connection setting:
driver, url, username, password

The *dialect* corresponding to the database vendor

Properties of Hibernate Configuration

- **Hibernate JDBC Properties**

Property	Description
connection.driver_class	It represents the JDBC driver class.
connection.url	It represents the JDBC URL.
connection.username	It represents the database username.
connection.password	It represents the database password.
connection.pool_size	It represents the maximum number of connections available in the connection pool.

- **Hibernate Configuration Properties**

Property	Description
hibernate.dialect	It represents the type of database used in hibernate to generate SQL statements for a particular relational database.
hibernate.show_sql	It is used to display the executed SQL statements to console.

Properties of Hibernate Configuration

- **Other Hibernate Properties**

Property	Description
hbm2ddl.auto	It automatically generates a schema in the database with the creation of SessionFactory: • create: the hibernate first drops the existing tables data and structure, then creates new tables and executes the operations on the newly created tables. • validate: hibernate only validates the table structure- whether the table and columns have existed or not. If the table doesn't exist then hibernate throws an exception. • update: Hibernate checks for the table and columns. If a table doesn't exist then it creates new tables and where as if a column doesn't exist it creates new columns for it • create-drop: Hibernate first checks for a table and do the necessary operations and finally drops the table after all the operations were completed.

Section 05

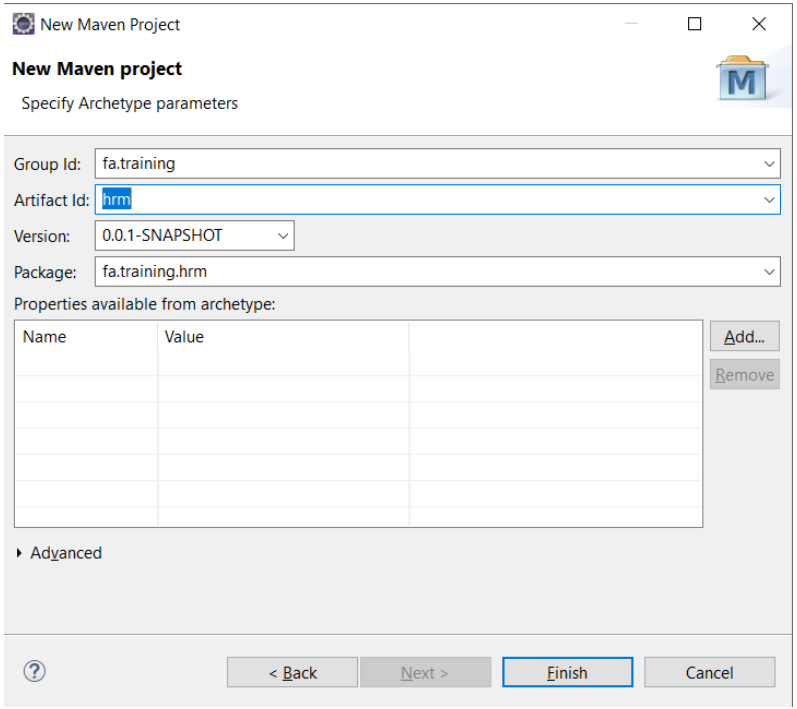
HIBERNATE FIRST EXAMPLE

Steps to create first Hibernate Application

1. Create the Java maven Project
2. Add all required dependencies for hibernate
3. Create the Persistent class
4. Create the mapping file for Persistent class
5. Create the Configuration file
6. Create the class that retrieves or stores the persistent object
7. Run the application

Create a Java Maven Project

- Create a new quickstart Maven project named "hrm":



Add hibernate dependency

```
<dependencies>
	<dependency>
		<groupId>org.hibernate</groupId>
		<artifactId>hibernate-core</artifactId>
		<version>5.4.12.Final</version>
	</dependency>
	<dependency>
		<groupId>com.microsoft.sqlserver</groupId>
		<artifactId>mssql-jdbc</artifactId>
		<version>7.4.1.jre8</version>
	</dependency>
</dependencies>
```

```
<build>
	<finalName>hrms</finalName>
	<plugins>
	<plugin>
		<groupId>org.apache.maven.plugins</groupId>
		<artifactId>maven-compiler-plugin</artifactId>
		<version>3.1</version>
		<configuration>
			<source>1.8</source>
			<target>1.8</target>
		</configuration>
	</plugin>
	</plugins>
</build>
```

Create the POJO /Persistent/Bean class

```java
@Entity
@Table(schema = "dbo", name = "Jobs")
public class Jobs {

    @Id
    @Column(name = "job_id", length = 10)
    private String jobId;

    @Column(name = "job_title", length = 255, nullable = false, unique = true)
    private String jobTitle;

    @Column(name = "min_salary", precision = 11, scale = 2)
    private double minSalary;

    @Column(name = "min_salary", precision = 11, scale = 2)
    private double maxSalary;

    public Jobs() {
    }

    public Jobs(String jobId, String jobTitle, double minSalary,
            double maxSalary) {
        super();
        this.jobId = jobId;
        this.jobTitle = jobTitle;
        this.minSalary = minSalary;
        this.maxSalary = maxSalary;
    }
//getter and setter methods
```

Create a Configuration file

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http:// sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<!-- Database connection settings -->
<property name="connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>
<property name="connection.url">jdbc:sqlserver://localhost:1433;databaseName=hrms</property>
<property name="connection.username">sa</property>
<property name="connection.password">12345678</property>

<!-- JDBC connection pool (use the built-in) -->
<property name="connection.pool_size">1</property>

<!-- SQL dialect -->
<property name="dialect">org. dialect.SQLServerDialect</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">update</property>

<mapping class="fa.training.entities.Jobs" />

</session-factory>
</hibernate-configuration>
```

Create a HibernateUtils class

```java
public class HibernateUtils {
    private static SessionFactory sessionFactory;

    static {
        // Create a new Configuration object
        Configuration cfg = new Configuration();
        cfg.configure();

        // Get the SessionFactory object from Configuration
        if (sessionFactory == null) {
            sessionFactory = cfg.buildSessionFactory();
        }

    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

}
```

Create a DAO class

```java
public class JobDaoImpl implements JobDao {

    @Override
    public boolean save(Jobs job) throws Exception {
        Session session = null;
        Transaction transaction = null;

        try {
            session = HibernateUtils.getSessionFactory().openSession();
            transaction = session.beginTransaction();

            Serializable result = session.save(job);

            transaction.commit();

            return (result != null);

        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
}
```

Create a Unit Test Script

- Create a UT Script to test the above DAO method

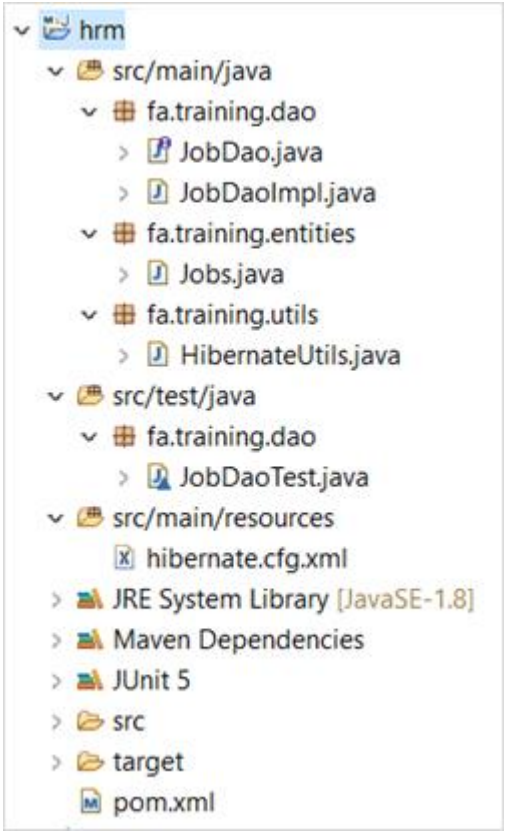
```
class JobDaoTest {
    static JobDao jobDao;

    @BeforeAll
    static void setUpBeforeClass() throws Exception {
        jobDao = new JobDaoImpl();
    }

    @Test
    void testSave1() throws Exception {
        Jobs job = new Jobs("J01", "Java Dev1", 1000, 2000);
        assertEquals(true, jobDao.save(job));
    }
    @Test
    void testSave2() throws Exception {
        Jobs job = new Jobs("J02", "Java Dev2", 1200, 2200);
        assertEquals(true, jobDao.save(job));
    }

}
```

Project Structure

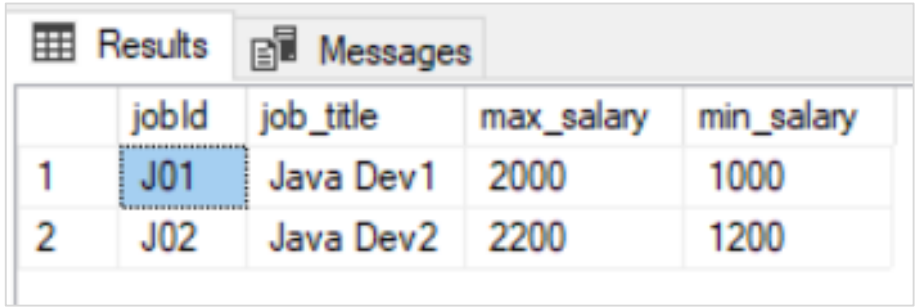


Results

```
Oct 04, 2020 12:06:22 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate ORM core version 5.4.12.Final
Oct 04, 2020 12:06:22 PM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>
INFO: HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
Oct 04, 2020 12:06:23 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production use!)
Oct 04, 2020 12:06:23 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001005: using driver [com.microsoft.sqlserver.jdbc.SQLServerDriver] at URL [jdbc:sqlserver://localhost;databaseName=hrms]
Oct 04, 2020 12:06:23 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {user=sa, password=****}
Oct 04, 2020 12:06:23 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
Oct 04, 2020 12:06:23 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 1 (min=1)
Oct 04, 2020 12:06:23 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.SQLServer2012Dialect
Oct 04, 2020 12:06:24 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess
[org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@773e2eb5] for (non-JTA) DDL
execution was not in auto-commit mode; the Connection 'local transaction' will be committed and the Connection will be set into auto-
commit mode.
Hibernate: create table dbo.Jobs (jobId varchar(255) not null, job_title varchar(255) not null, max_salary double precision, min_salary
double precision, primary key (jobId))
Hibernate: alter table dbo.Jobs drop constraint UK_hiqsn2scso4em9j18jb24ioo8
Hibernate: alter table dbo.Jobs add constraint UK_hiqsn2scso4em9j18jb24ioo8 unique (job_title)
Oct 04, 2020 12:06:25 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Hibernate: insert into dbo.Jobs (job_title, max_salary, min_salary, jobId) values (?, ?, ?, ?)
Hibernate: insert into dbo.Jobs (job_title, max_salary, min_salary, jobId) values (?, ?, ?, ?)
```

Results

- The result after selecting Jobs table.



	jobId	job_title	max_salary	min_salary
1	J01	Java Dev1	2000	1000
2	J02	Java Dev2	2200	1200

Summary

- Hibernate Overview
- Hibernate Features
- Hibernate Architecture
- Configuration
- Hibernate First Example

THANK YOU!

Q & A