Hibernate - II

Introduction to Hibernate

Agenda

1 **Introduction to Hibernate**

Introduction to Hibernate

Objectives

In this module you will be able to:

- Write a Simple Hibernate program
- Understand the Architecture
- Use Hibernate API

A simple Hibernate program

Let us look at an example and get the feel of Hibernate before understanding the underlying concepts

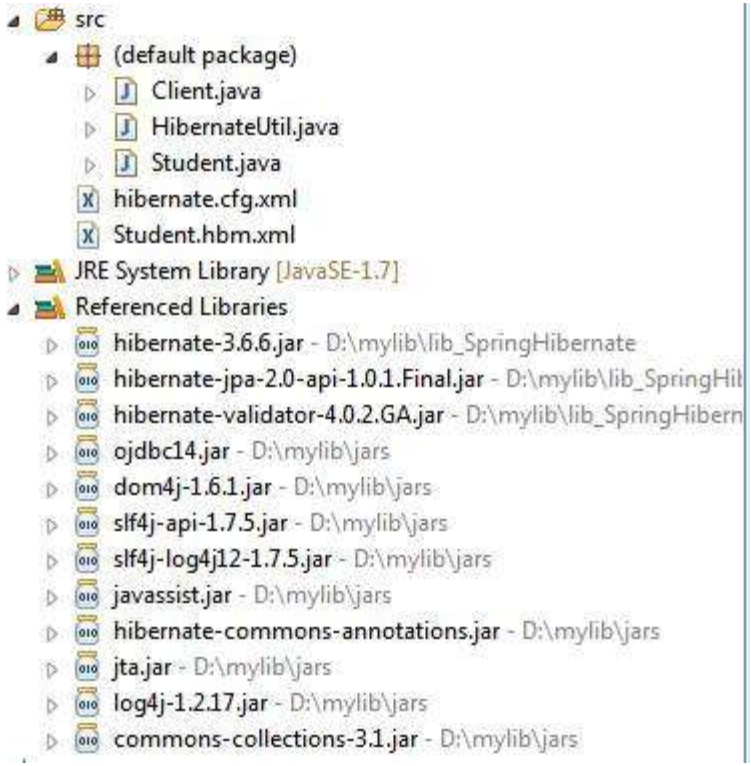
- Create a table Student in Oracle SQL Plus

```
create table student(
sid number(5) not null primary key,
sname varchar(15)
);
```

- Create a Java Application Project in Eclipse

A simple Hibernate program (Contd.).

- Add the following jar files to the Build path



A simple Hibernate program (Contd.).

- Create a persistent class , POJO **Student.java**

```
public class Student {
    private int id;
    private String name;

    public Student() {
    }

    public int getId() {
        return id;
    }
    private void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Properties of Student

No argument constructor

Follows Java bean style

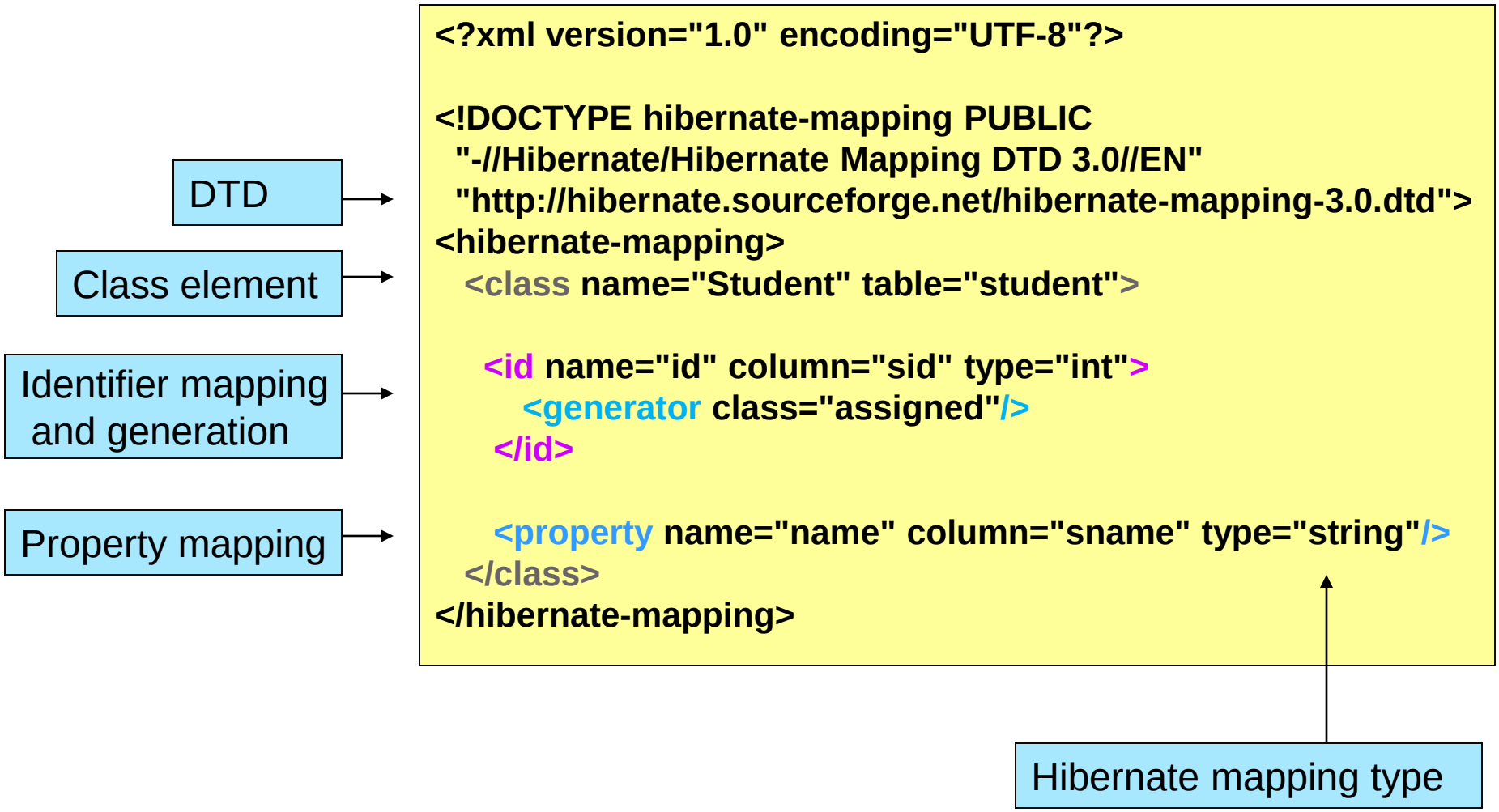
The term POJO refers to normal Java objects that does not serve any other special role or implement any special interfaces of any of the Java frameworks (EJB, JDBC, DAO, JDO, etc...).

A simple Hibernate program (Contd.).

- Place this Java class in default package. In this example a POJO (plain old Java object) is used to represent the data in the table created in Oracle database.
- The Java class specifies the fields for the columns in the table and uses simple setters and getters to retrieve and write the data.
- A class whose instances of which can be persisted is called Persistent class.
- The class follows Java Bean style of writing the code.
- The class has all the properties declared as private. Properties will be mapped to columns in the mapping file.
- The class has a no argument constructor, which we should have compulsorily as Hibernate uses reflection to create instances.
- Then we have setter (mutator) and getter (accessor) methods for each of the properties in the class.
- The setId method can be made private as we do not change the identification of the object and it can be generated automatically by hibernate.

A simple Hibernate program (Contd.).

- Create an XML hibernate mapping definition file **Student.hbm.xml**



A simple Hibernate program (Contd.).

- We will discuss in detail about the mapping file contents in module “Basic O/R Mapping”. Mapping file tells Hibernate which tables and columns to use to load and store objects. Each persistent class has a corresponding database table. To map POJOs to tables a Hibernate mapping file is used. The student objects are called “Persistent Objects” or “Entities” for they can be persisted in database and they represent real world entities.
- Object/relational mappings are defined in an XML document. The mapping document is designed to be readable and hand-editable. The mapping language is Java-centric, meaning that mappings are constructed around persistent class declarations, and not table declarations.
- The naming convention for mapping files is to use the name of the persistent class with the hbm.xml extension. The mapping file for the Student class is thus **Student.hbm.xml**
- *class* tag
- name attribute - requires the name of the POJO class
- table attribute - requires the name of the corresponding table in the database
- *id* tag- Each persistent object must have an identifier. It is used by Hibernate to identify that object uniquely. Here we choose the id as identifier of a Student object. Used to provide the algorithm for generating the key values. The **id element** also describes how the key value is generated.
- *property* tags are used to provide names of the properties of the persistent class, their data types and the name of the column to which the property should map onto.

A simple Hibernate program (Contd.).

- Create an XML configuration file **hibernate.cfg.xml** which provides Hibernate with all the details needed to connect to the database

```
<?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>
   <!-- Oracle dialect -->
    <property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
   <!-- Database connection settings -->
    <property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:oracle1</property>
    <property name="hibernate.connection.username">scott</property>
    <property name="hibernate.connection.password">tiger</property>
   <!-- Echo all executed SQL to stdout -->
    <property name="hibernate.show_sql">true</property>
   <!-- Enable Hibernate's automatic session context management -->
    <property name="hibernate.current_session_context_class">thread</property>
   <!– Mapping file entry -->
    <mapping resource="Student.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

A simple Hibernate program (Contd.).

- To keep things simple, a single XML configuration file for Hibernate is used that contains all the configuration details. It is recommended that you give your configuration file the default name hibernate.cfg.xml and place it directly in the source directory of your project, outside of any package. That way, it will end up in the root of your classpath after compilation, and Hibernate will find it automatically. The configuration file provides details of the **driver class**, **Driver URL**, **username, password** to connect to the database. It also provides details about the connection pooling class, logging and also about the mapping files for the persistent classes.
- hibernate.dialect :The classname of a Hibernate org.hibernate.dialect.Dialect which allows Hibernate to generate SQL optimized for a particular relational database. **eg.** ull.classname.of.Dialect. In most cases Hibernate will actually be able to choose the correct org.hibernate.dialect.Dialect implementation to use based on the JDBC metadata returned by the JDBC driver.
- **<!-- JDBC connection pool (use the built-in) -->**
- **<property name="connection.pool_size">1</property>**

- The hibernate.current_session_context_class configuration parameter defines which org.hibernate.context.CurrentSessionContext implementation should be used and we bind it to the value “thread”.

A simple Hibernate program (Contd.).

Create a utility class to create SessionFactory. We should have **only one SessionFactory per Database**.

We use the **SessionFacatory to create sessions** through which **we do the transactions with the database**.

- SessionFactory will read the configuration files like **Student.hbm.xml** and **hibernate.cfg.xml** through which it gets the necessary information to work with the Database.

A simple Hibernate program (Contd.).

- Create **HibernateUtil.java** file, a hibernate utility class with a convenient method to get Session factory object

```
import org.hibernate.cfg.Configuration;
import org.hibernate.SessionFactory;
public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            // Create the SessionFactory from standard (hibernate.cfg.xml)  config file
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Log the exception
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

A simple Hibernate program (Contd.).

- Create **Client.java** file and run your project to insert record into Student table

```
import java.util.*;
import org.hibernate.*;
public class Client {
   public static void main(String[] args) {
      // Create Session object
      Session session = HibernateUtil.getSessionFactory().openSession();

      // Perform life-cycle operations under a transaction
      Transaction tx = null;
      try {
         tx = session.beginTransaction();

         // Create a Student1 object and save it
         Student1 s1 = new Student1();
         s1.setId(33);
         s1.setName("Asma");
         session.save(s1);

         tx.commit();
      } catch (HibernateException e) {
                e.printStackTrace();
      } finally {
         session.close();
      }
   }
}
```

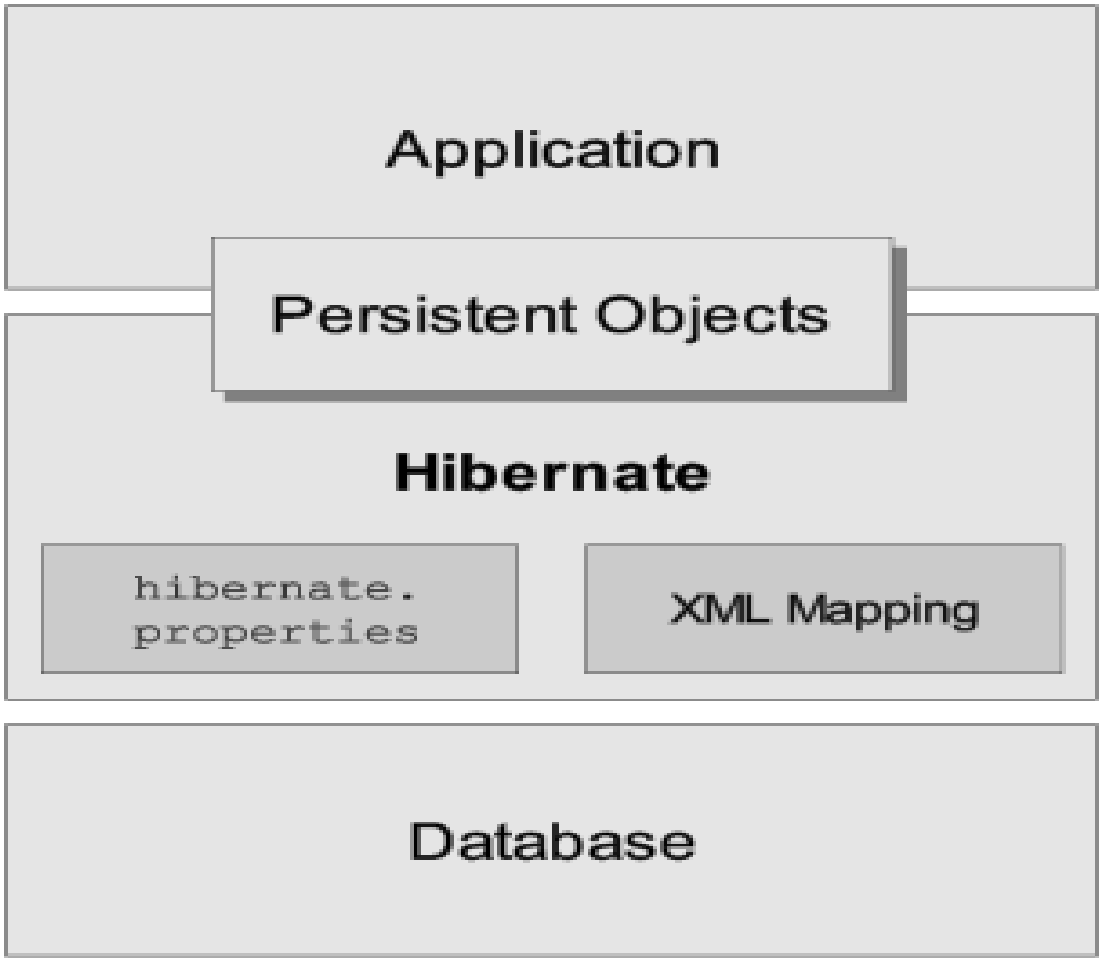
A simple Hibernate program (Contd.).

- It is very easy to create an object and insert records into the table. Look at the code in the previous slide; there are no statements present to connect to the database. All that information is provided in the configuration file and done behind the scenes.

- Create an instance of the Persistent class Student, provide the details of the properties through the setter methods and pass the instance to the save method of the session. This statement will insert a record with the values of the properties in the Student object.
- **Session object is used to demarcate the transactions.**

Architecture

Now it will be easier to understand the Hibernate architecture.



Hibernate uses persistent objects (POJOs) along with XML mapping documents for persisting objects to the database layer

Hibernate uses database and configuration data to provide persistence services (and persistent objects) to the application

Hibernate API

org.hibernate.cfg.Configuration

org.hibernate.SessionFactory

org.hibernate.Session

org.hibernate.Transaction

org.hibernate.Query

Hibernate API – The Configuration class

- Hibernate configuration
  - The Configuration object needs to know about all XML mapping files before you build the SessionFactory
  - Represents a set of mapping files specified in the Hibernate configuration file

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

→ Hibernate searches for file **hibernate.cfg.xml** in the root of the classpath.
Loads it if found else An exception is thrown if not found

The Hibernate configuration file

- All the configuration information is stored in an xml file named hibernate.cfg.xml
- Alternatively the configuration information can also be stored in hibernate.properties file
- Different categories of information is stored in the configuration file
- Following are few of them:
  - Database dialect and connection
  - Database driver (Driver class, Driver URL)
  - Connection pool
  - Caching
  - Transaction
  - Logging
  - Data Source

The Hibernate configuration file

- **Database Dialect**
  - Based on your requirement you use different Databases like MySQL, PostgreSQL, Oracle etc. Hibernate supports a lot of dialects for different databases and their corresponding versions.
  - The value specified with this property makes hibernate generate appropriate SQL query for a particular database. Hibernate uses this configuration to know which database is being used, so that if necessary, it can generate database specific code.

- **UserName and Password**
  - Connection information like the username and password should also be provided in this file.

- **Database Driver**
  - Information like name of the driver class, driver URL for the database connection will be mentioned here.

The Hibernate configuration file

- **Connection Pool**
  - By default Hibernate provided connection pooling. But the hibernate documentation suggests that we use a third party utility for a production environment. We can use C3P0, Proxool connection pooling facility
- **Caching**
  - Hibernate provides 2 levels of caching for better performance, we can switch on or switch off the second level cache in the configuration file.
- **Transaction**
  - Here we can mention the different kind of transaction contexts like JTA, JDBC.
- **Logging**
  - Hibernate uses log4j for logging, which can be enabled here.
- **Data Source**
  - JNDI information for connecting to a Database through a managed environment is provided here.

The SessionFactory interface

Initialize Hibernate by building a SessionFactory object from a Configuration object

A SessionFactory is an object representing a particular Hibernate configuration for a particular set of mapping metadata

An Hibernate startup procedure using automatic configuration file detection:

```
SessionFactory sessionFactory = new
                Configuration().configure().buildSessionFactory();
```

In most Hibernate applications, the SessionFactory should be instantiated once during application initialization

A Session is created using SessionFactory

The SessionFactory is thread-safe and can be shared among application threads

The SessionFactory interface

Hibernate applications use an XML configuration file, that contains all the required configuration details.

It is recommended to give this configuration file the default name hibernate.cfg.xml and place it directly in the src directory of your project, outside of any package. That way, it will end up in the root of your classpath after compilation and Hibernate will find it automatically.

In most Hibernate applications, the SessionFactory should be instantiated once during application initialization. The single instance should then be used by all code in a particular process and the session object should be created using this single SessionFactory. The SessionFactory is thread-safe and can be shared; a Session is a single-threaded object.

The Session Interface

The session object acts as an interface between your application and the data that you store in the database. This object acts as a factory for creating Transaction, query and criteria objects.

The Session interface(available in the package org.hibernate) provides methods to insert, update, delete and retrieve the entity object.

The Session Interface (Contd.).

The Session object is obtained from a SessionFactory instance

```
sessionFactory = new Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();
```

Factory for Transaction – specifies transaction boundaries

```
Transaction tx = session.beginTransaction();
```

Referred as a unit of work

- To begin a unit of work you open a Session
- To end a unit of work you close a Session

Responsible for storing and retrieving objects

```
session.save( a persistence object );
```

The Session Interface

The session is bound to the thread behind the scenes.

To enable the thread-bound strategy in your Hibernate configuration file, s*et the value of the property*

hibernate.current_session_context_class *to* ***thread***

Hibernate framework automatically generates the query depending on the methods called on the session object.

To display this auto generated query on the console, set the value of the property

hibernate.show_sql to **true**

Summary

In this module you will be able to:

- Write a Simple Hibernate program
- Understand the Architecture
- Use Hibernate API

Thank You