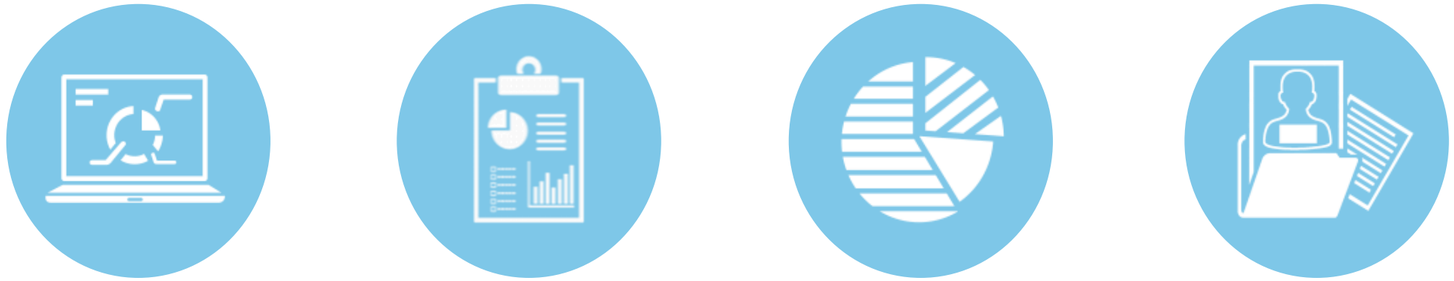
Advanced Java

Lesson 5—Java Hibernate



Learning Objectives

- Discuss Hibernate basics
- Explain Hibernate ORM feature
- Set up a Project with Hibernate
- Describe Hibernate Annotations
- Explain Hibernate CRUD Operation

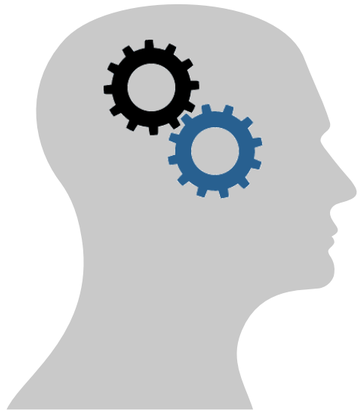
Advanced Java

Topic 1—Hibernate Basics

Getting Started

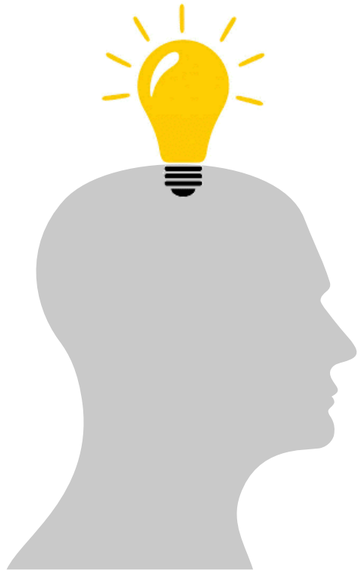
- Hibernate Framework was built by **Gavin King** in 2001
- The latest version of Hibernate is **5.2.10**

Think Through!



Let's start by discussing why we need Hibernate and why it was created?

Think Through!

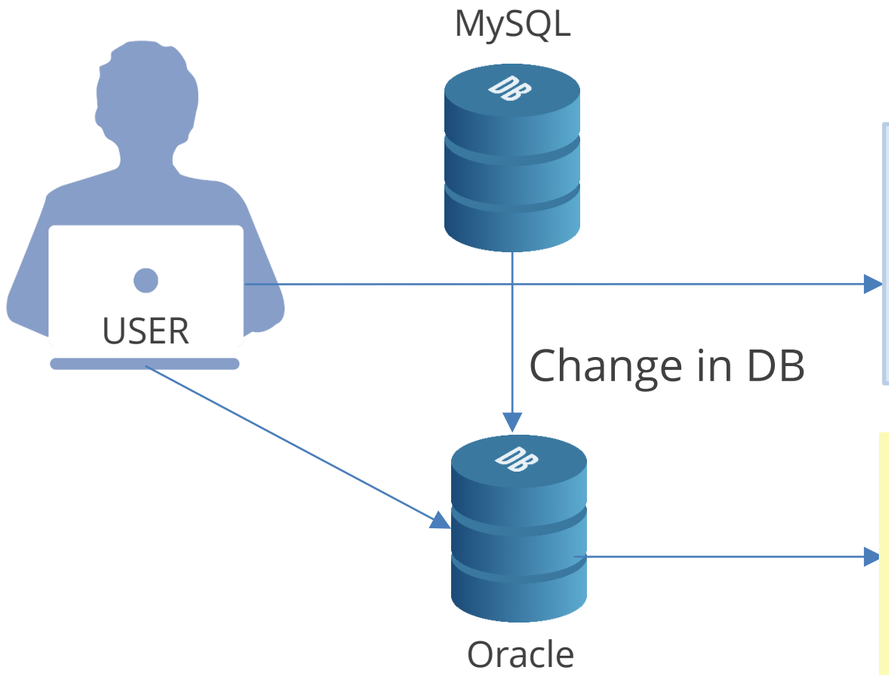


Hibernate is a framework that simplifies the development of Java application to interact with the database.

Lets understand it why we need it with the help of two scenarios.

Scenario 1: Changing Database

Let’s consider a scenario where a user has a program with MySQL database connectivity. The user needs to write MySQL specific code to create database connection.



Database Connectivity Code (MySQL specific)
+
Business Logic

Database Connectivity Code (Oracle specific)
+
Business Logic

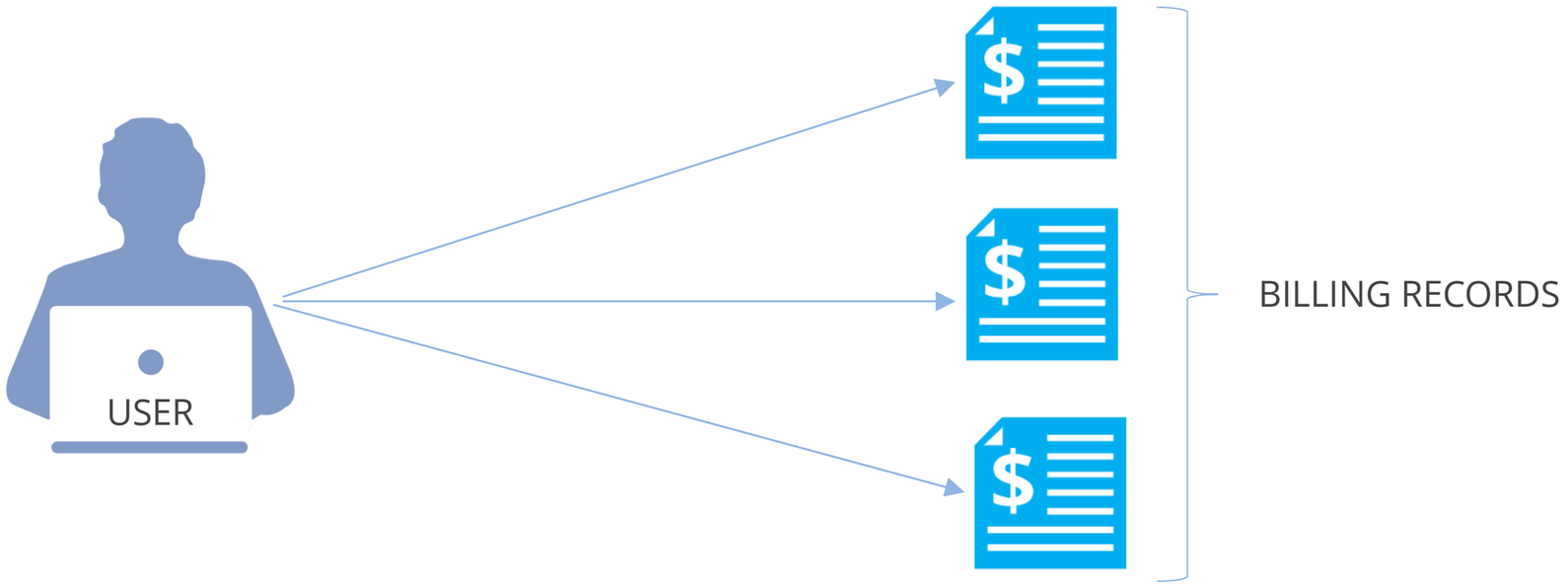
The user has to change the codes and queries to map them to Oracle database

Hibernate converts database-specific queries automatically, based on dialect provided. It uses HQL (hibernate Query Language), which is independent of Database.

Scenario 2: Billing Record

Let’s consider a scenario of a user with various billing records.

The USER and the BILLING RECORDS data needs to be stored in DBMS. For this, the user class has to be mapped to user table, and the Billing Records has to be mapped to the BillingRecords table.



Scenario 2: Billing Record

CLASS REPRESENTATION

Creating the SQL schema for USER and BILLINGRECORDS classes

```
class User
{
private int id;
private String name;
private Set billingrecords
// getter method
//setter method
}
```

```
create table USER
{
id number (3) primary key not
null,
name varchar(50) not null,
address varchar(40),
age number (3) ,
}
```

```
class BillingRecords
{
private int accountNo;
private String accountType;
private User user;
// getter method
//setter method
}
```

```
create table BILLINGRECORDS
{
accountnumber number (10) primary key not
null,
acccounttype varchar (6) not null,
……
id number(3) foreign key references user
}
```

Scenario 2: Billing Record

OBJECT RELATIONAL MISMATCH PROBLEMS

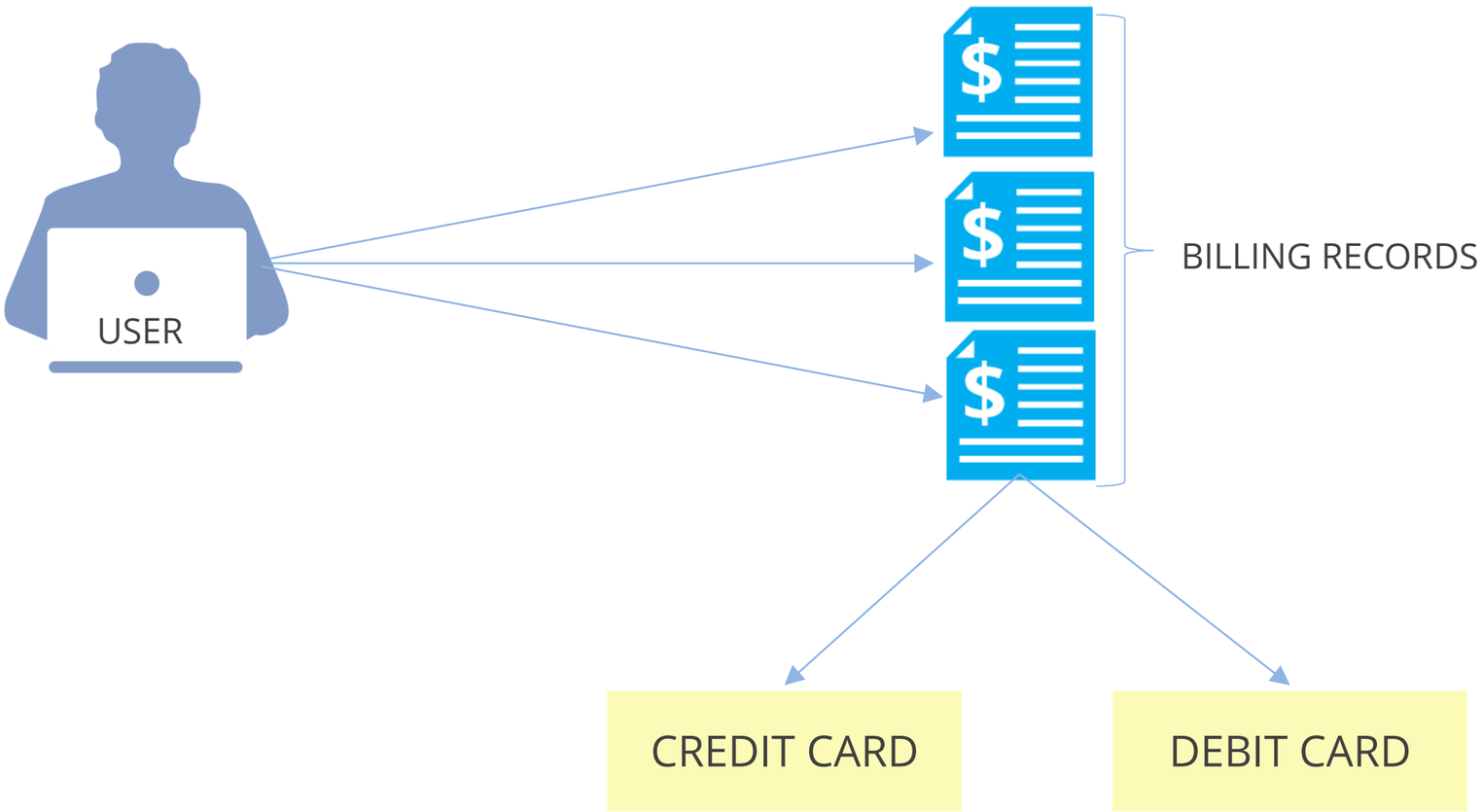
What could be the possible problems?

- Association: Java can use association by having another class variable as member; in SQL, no standard concept is available to represent association
- Navigating data: It is different in Java and SQL
- Cost mismatching due to manual handling of object/relational mismatching

Scenario 2: Billing Record

ADDING SUBCLASSES

Let's add two subclasses CREDITCARD and DEBITCARD to the records:



Scenario 2: Billing Record

ADDING SUBCLASSES: RELATED PROBLEMS

What could be the possible problems?

- Structured Query Language (SQL) does not support supertable-subtable relationship
- Table is not a ‘type,’ so it is difficult to create supertable-subtable relationship
- Writing SQL query for polymorphic relation is difficult

Scenario 2: Billing Record

ADDING SURROGATE KEY COLUMN

- In USER table, id is the primary key. This is a foreign key for BILLINGRECORD table.
- It is difficult to change id; we need to update not only the id column in USER, but also the foreign key column in BILLINGRECORD
- Add surrogate key in USER and BILLINGRECORD table
- Surrogate key isn't presented to the user and is only used for identification of data inside the software system

Scenario 2: Billing Record

ADDING SURROGATE KEY COLUMN: RELATED PROBLEMS

- user_surroagte_id and account_number_id are system generated values. It is difficult to decide whether these columns should be added to data model or not.
- Different persistence solutions have chosen different strategies. This can cause confusion.

What is the Solution?

Hibernate provides Object Relational Mapping, which takes care of these issues.

Advanced Java

Topic 2—ORM and its Features

ORM

ORM refers to the automated (and transparent) persistence of objects in a Java application to the tables in a relational database, using metadata that describes the mapping between the objects and the database.

Java class

User Table

It has two members ids and names

```
class User
{
private int id;
private String name;
//getter methods
//setter methods
}
```

id	name

It has two column ids and names.

Column id and name of user table are representing members of User class id and name respectively.

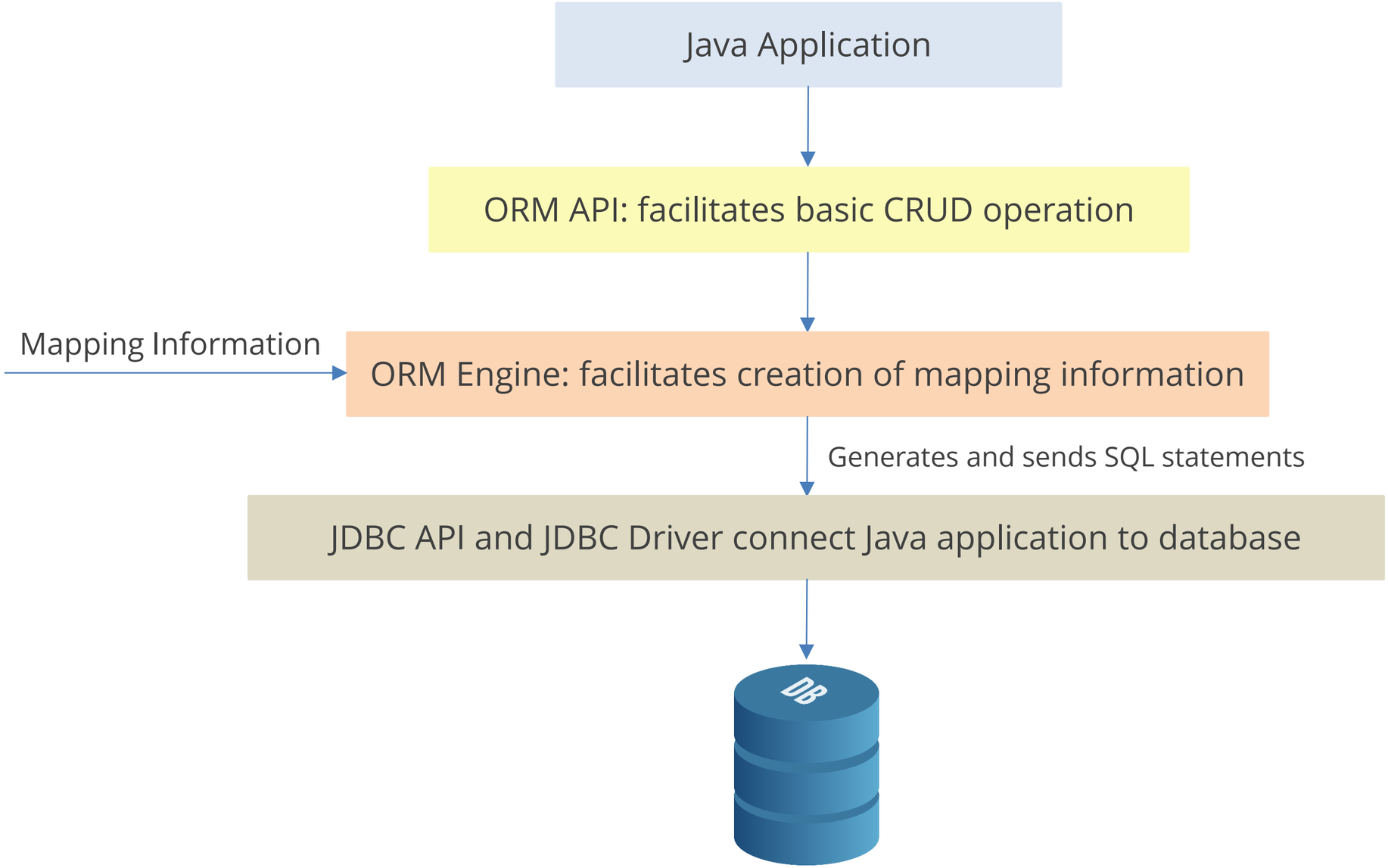
Advantages of using ORM

- It can significantly reduce the development time
- It involves writing less code
- It increases system performance
- You can support different database management systems by adopting ORM. It provides portability

ORM: Features

- By mapping between logical business model and physical storage model, ORM implements domain model pattern
- It navigates object relational transparency
- ORM provides one way to translate query to appropriate database syntax
- It provides concurrency support
- It provides a host of services that enable developers to focus on business logics
- It provides cache management to reduce load on database
- Surrogate key, Identifier, and other key features can be automated in ORM

ORM Architecture



Advanced Java

Topic 3—Hibernate as an ORM Framework

Hibernate as an ORM Framework

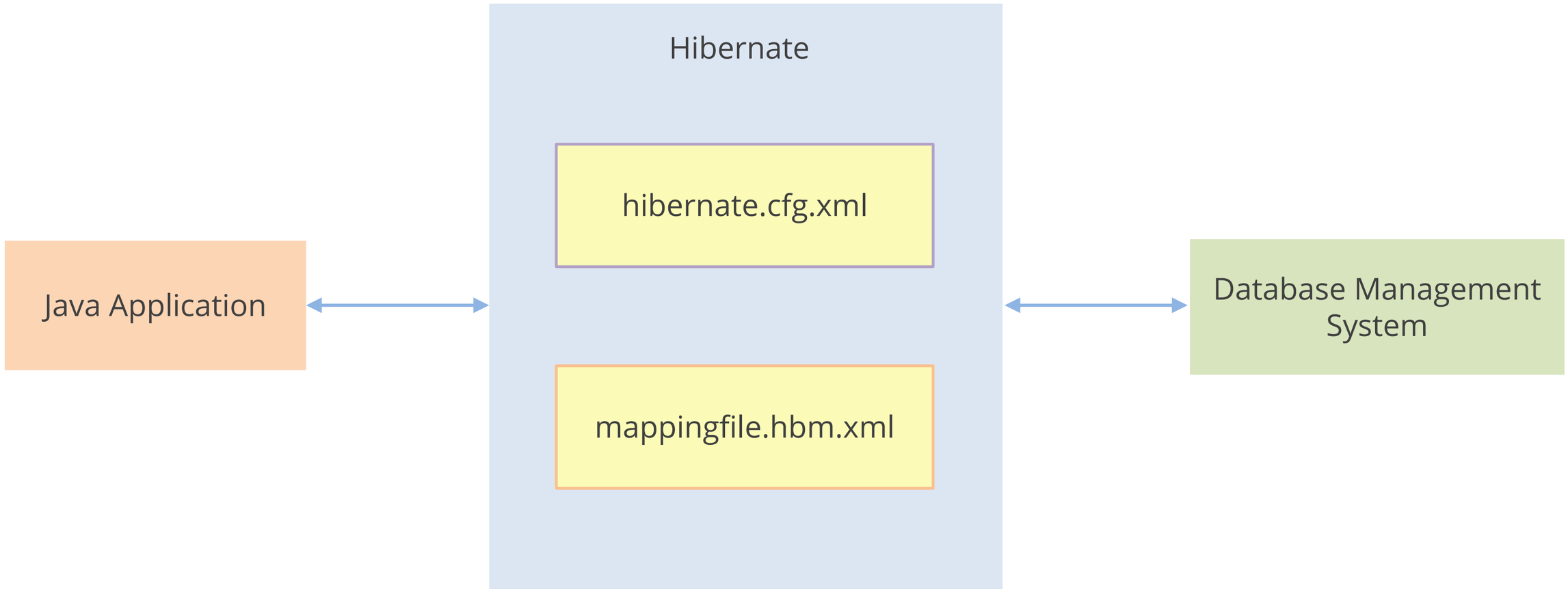
"As an Object/Relational Mapping (ORM) framework, Hibernate is concerned with data persistence as it applies to relational databases (via JDBC)."

Reference : http://hibernate.org/orm/

Hibernate Configuration Files

DBMS-specific details and mapping file details are specified in hibernate.cfg.xml and mappingfile.xml.

Hibernate engine uses these files to generate DBMS-specific SQL syntax.



Features of Hibernate ORM

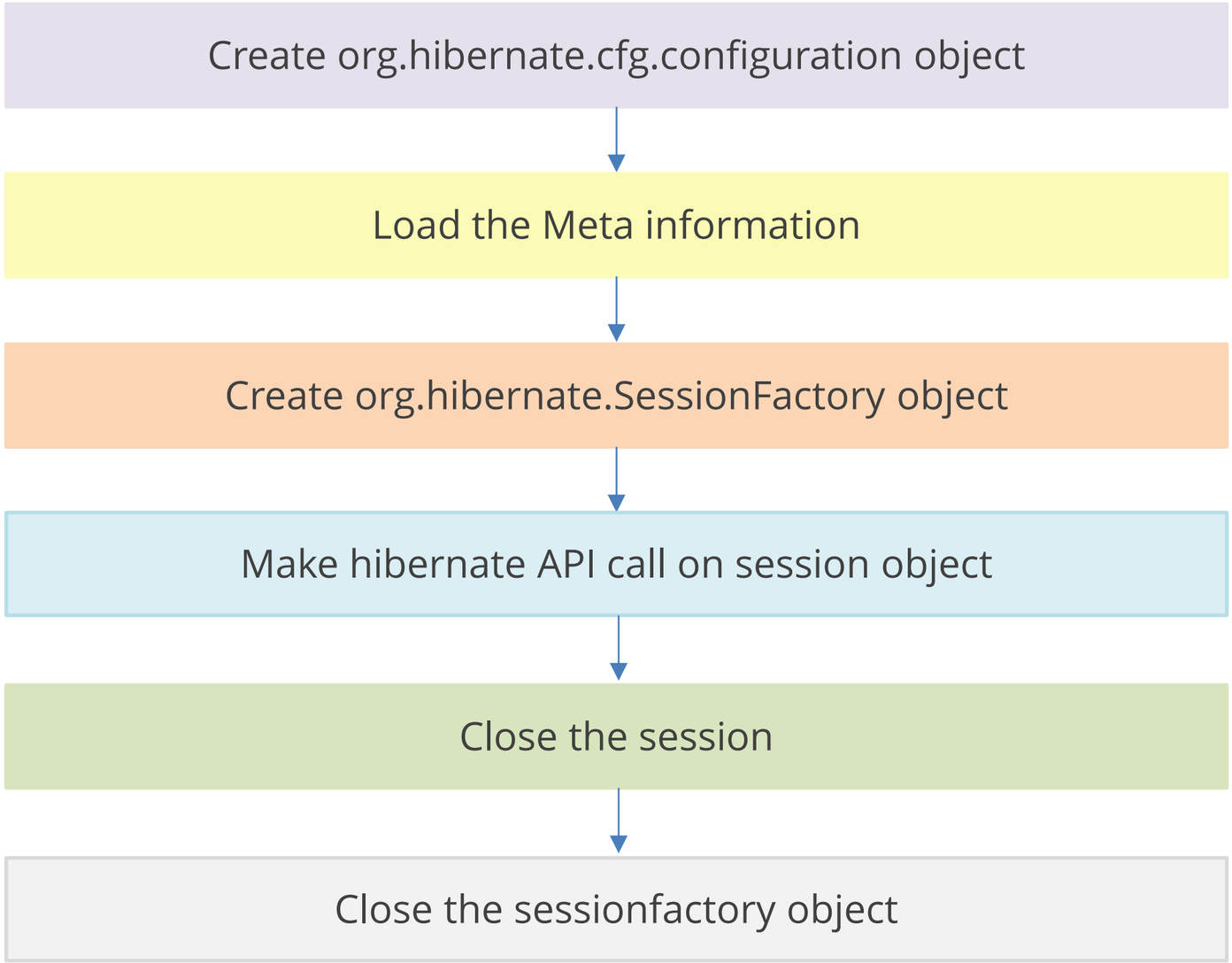
- Hibernate ORM provides its own native API, in addition to full **JPA (Java Persistence API)** supports.
- It maps Java POJO's (Plain Old Java Object) to relational database.
- It provides rich tool set.
- Performance: Fetch strategies, caching, byte code enhancement
- It is part of JBoss community.

You can download the latest version of Hibernate from: http://www.hibernate.org/downloads

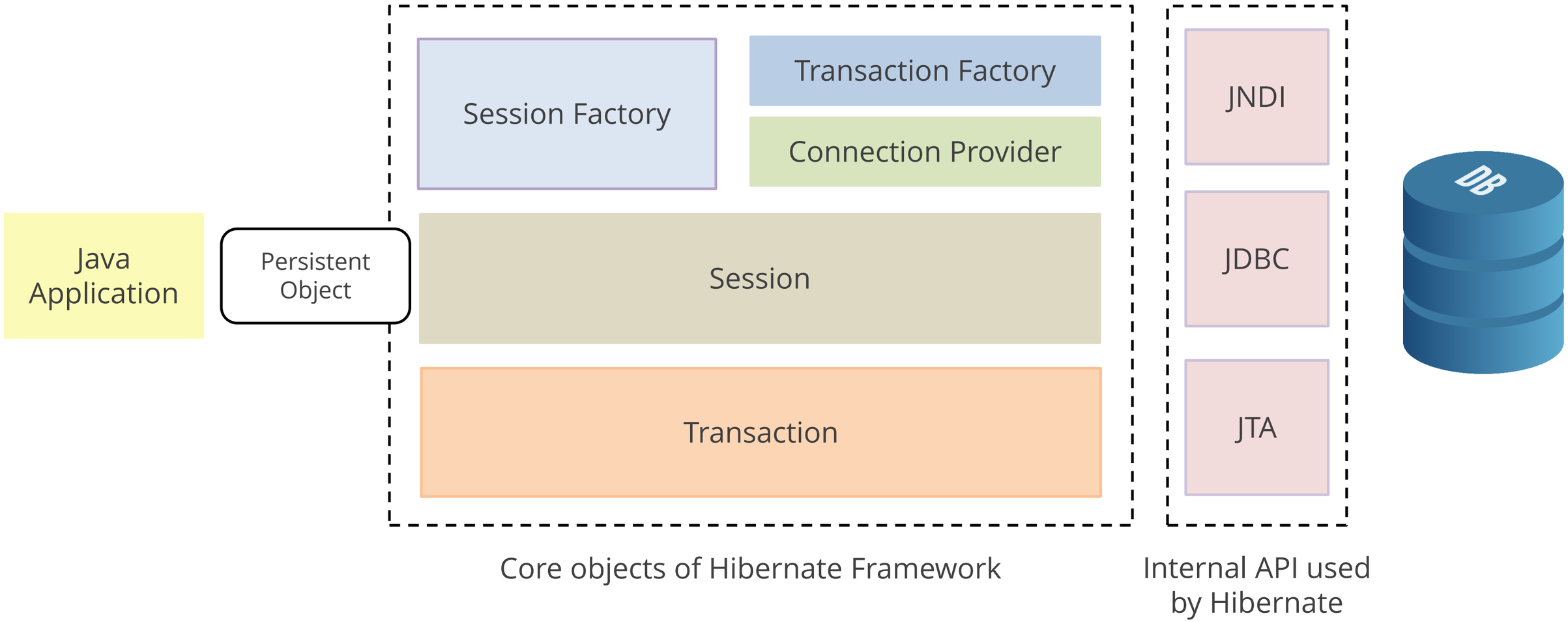
JPA (Java Persistence Application Programming Interface)

- Java EE 5.0 platform provides a standard persistence API named JPA. As part of the EJB 3.0 specification effort, it is supported by all major vendors of the Java industry.
- Hibernate model implements Java Persistence object relational API.
- Hibernate persistence provider can be used in any environment of Java platform, Java SE or Java EE.

Hibernate Programming Model



Hibernate Architecture



Persistent Object

Session object is created within the Database Layer in every DAO method. It is known as persistence object.

It has three lifecycle states:

1. Transient: An object is transient if it has just been instantiated using the new operator, and it is not associated with a Hibernate Session.

2. Persistent: 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.

3. Detached: 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.

Hibernate Core APIs

1. Configuration (org.hibernate.cfg): It gives two services in hibernate application:

    - Loads mapping file and configuration file into memory and makes them available to hibernate engine
    - Acts as factory to create the SessionFactory

2. SessionFactory (org.hibrnate.SessionFactory)

    - Hibernate (engine) implements SessionFactory interface
    - SessionFactory is one per DBMS (mostly one per application)
    - SessionFactory is Thread safe
    - SessionFactory is not a Singleton
    - It creates Session object, SessionFactory encapsulates, second level cache, connection pool, meta information cache, and pool of session

Hibernate Interface

3. Session (org.hibernate.Session)

- Hibernate engine implements Session interface
- Session object acts as persistent manager
- Session object is a light weight object
- It encapsulates connection and first-level cache
- Session object is not thread-safe
- In every dao method, Session object is created
- It is used for CRDD operation

4. Transaction (org.hibernate.Transaction)

- Hibernate engine implements this interface
- When database connection is created by hibernate, connection associated with session is in autocommit disable mode.
- Whenever any CRUD operation is performed, the changes will not be reflected in the database unless connection is maintained in auto-commit enabled mode.

Hibernate Interface

IdentifierGenerator

All the generator classes implement the **org.hibernate.id.IdentifierGenerator interface**.

The application programmer may create one's own generator classes by implementing the IdentifierGenerator interface.

Hibernate framework provides many built-in generator classes:

- assigned: Default, value has to be explicitly assigned to persistent object before persisting it. <generator class = assigned/>
- increment: It increments value by 1. It generates **short**, **int,** or **long** type identifier.
- native: It uses identity, sequence, or hilo, depending on the database vendor.
- sequence: It uses the sequence of the database. If there is no sequence defined, it creates a sequence automatically. For example, in case of Oracle database, it creates a sequence named HIBERNATE_SEQUENCE.
- hilo: It uses high and low algorithm to generate the id of type short, int, and long.
- identify: It is used in Sybase, My SQL, MS SQL Server, DB2, and SQL to support the id column. The returned id is of type short, int, or long.

Advanced Java

Topic 4—Setting up a Project with Hibernate

Setting Up a Project with Hibernate

1. Create a Java project and add required jars from the required folder of hibernate downloads
2. Add ojdbc14.jar or mysqlconnector.jar depending on the SQL vendor you are connecting to
3. Create the POJO class and save it in src folder
4. Create the hibernate.cfg.xml file in src folder(configuration file) with specific RDBMS dialect

RDBMS	Dialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle9i	org.hibernate.dialect.Oracle9iDialect
Oracle10g	org.hibernate.dialect.Oracle10gDialect
MySQL	org.hibernate.dialect.MySQLDialect
DB2	org.hibernate.dialect.DB2Dialect

5. Create the ClassName.hbm.xml file in src folder (mapping file)
6. Create the class that retrieves or stores the persistent object and save it in src folder
7. Run the application

Setting Up a Project with Hibernate: Step 1

Create a Java project and add required jars from the required folder of hibernate downloads

Add ojdbc14.jar or mysqlconnector.jar depending on the SQL vendor you are connected to

Create the POJO class and save it in src folder

Create the hibernate.cfg.xml file in src folder (configuration file)

Create the ClassName.hbm.xml file in src folder (mapping file)

Create the class that retrieves or stores the persistent object and save it in src folder

Run the application

1. Download the latest version of Hibernate from http://hibernate.org/orm/downloads/
2. Right click on project
3. Click on build path
4. Click on add External JARs
5. Add all required Hibernate JARs

Setting Up a Project with Hibernate: Step 2

Create a Java project and add required jars from the required folder of hibernate downloads

Add ojdbc14.jar or mysqlconnector.jar depending on the SQL vendor you are connected to

Create the POJO class and save it in src folder

Create the hibernate.cfg.xml file in src folder (configuration file)

Create the ClassName.hbm.xml file in src folder (mapping file)

Create the class that retrieves or stores the persistent object and save it in src folder

Run the application

Download mysqlconnector.jar from:

https://dev.mysql.com/downloads/connector/j/

Download link for ojdbc14.jar for Oracle 11g:

http://www.java2s.com/Code/Jar/o/Downloadojdbc14jar.htm

Setting Up a Project with Hibernate: Step 3

- Create a Java project and add required jars from the required folder of hibernate downloads
- Add ojdbc14.jar or mysqlconnector.jar depending on the SQL vendor you are connected to
- Create the POJO class and save it in src folder
- Create the hibernate.cfg.xml file in src folder (configuration file)
- Create the ClassName.hbm.xml file in src folder.(mapping file)
- Create the class that retrieves or stores the persistent object and save it in src folder
- Run the application

```
class Student
{
private int rollNo ;
private String name;
// setter menthods
//getter methods
}
```

Student is a POJO class containing two member variables, rollNo and name.

It uses setter methods to set the values for member variables and getter methods to fetch the values.

Setting Up a Project with Hibernate: Step 4

- Create a Java project and add required jars from the required folder of hibernate downloads
- Add ojdbc14.jar or mysqlconnector.jar depending on the SQL vendor you are connected to
- Create the POJO class and save it in src folder
- Create the hibernate.cfg.xml file in src folder (configuration file)
- Create the ClassName.hbm.xml file in src folder (mapping file)
- Create the class that retrieves or stores the persistent object and save it in src folder
- Run the application

Hibernate configuration file configures class Student to table studentname using <hibernate-mapping> tag. Hibernate.cfg.xml is used to mention database detail.

```
<?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>
                <property name="dialect">org.hibernate.dialect.MySQLDialect
</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/databasename</pro
perty>
        <property name="connection.username">username</property>
        <property name="connection.password">password</property>
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
 <property name="hbm2ddl.auto">create</property>
    <mapping resource="Student.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

Setting Up a Project with Hibernate: Step 5

- Create a Java project and add required jars from the required folder of hibernate downloads
- Add ojdbc14.jar or mysqlconnector.jar depending on the SQL vendor you are connected to
- Create the POJO class and save it in src folder
- Create the hibernate.cfg.xml file in src folder (configuration file)
- Create the ClassName.hbm.xml file in src folder (mapping file)
- Create the class that retrieves or stores the persistent object and save it in src folder
- Run the application

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

 <hibernate-mapping>
  <class name="Student" table="studenttable">
    <id name="id">

    </id>

    <property name="name"></property>


  </class>

 </hibernate-mapping>
```

POJO class name is student. The mapping file name would be Student.hbm.xml which maps student class to student table.

Setting Up a Project with Hibernate: Step 6

- Create a Java project and add required jars from the required folder of hibernate downloads
- Add ojdbc14.jar or mysqlconnector.jar depending on the SQL vendor you are connected to
- Create the POJO class and save it in src folder
- Create the hibernate.cfg.xml file in src folder (configuration file)
- Create the ClassName.hbm.xml file in src folder (mapping file)
- Create the class that retrieves or stores the persistent object and save it in src folder
- Run the application

Creating configuration object and hibernate.cfg.xml file:

```
public class Client
{
StandardServiceRegistry standardRegistry = new

StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build
();

Metadata metaData = new
MetadataSources(standardRegistry).getMetadataBuilder().build();
 sessionFactory = metaData.getSessionFactoryBuilder().build();

}
```

Setting Up a Project with Hibernate: Step 7

Create a Java project and add required jars from the required folder of hibernate downloads

Add ojdbc14.jar or mysqlconnector.jar depending on the SQL vendor you are connected to

Create the POJO class and save it in src folder

Create the hibernate.cfg.xml file in src folder (configuration file)

Create the ClassName.hbm.xml file in src folder (mapping file)

Create the class that retrieves or stores the persistent object and save it in src folder

Run the application

Use <property name="hbm2ddl.auto">update</property> to avoid creating studenttable manually.

Hibernate will automatically create studenttable.

Run the program to map the:

1. Student class to Student table.
2. int rollNo (member of class Student) to column rollNo
3. String name (member of class Student) to column name

Output:

```
Hibernate: drop table if exists studenttable
Hibernate: create table studenttable (rollNo integer not
null,name varchar(5) );
```

Advanced Java

Topic 5—Hibernate Annotations

Hibernate Annotation

You have Learned how Hibernate uses XML mapping file for the transformation of data from POJO to database tables and vice versa

Hibernate annotation is the newest way to define mappings without the use of XML file. You can use annotations in addition to or as a replacement of XML mapping metadata.

Creating POJO Class With Annotation

```
@Entity // Every persistent POJO class is an entity and is declared using the @Entity
annotation (at the class level):

@Table // @Table annotation, hibernate will use the class name as the table name by default

public class Student
 {
@Id  // @Id declares the identifier property of this entity.
    public int  id ;

private String name; // @Column annotation specifies the details of the column for this
property or field.
If // if it is  is not specified, property name will be used as the column name by default.

// setter and getter methods
}
```

Use POJO class student, hibernate.cfg.xml file, and annotations instead of hibernate mapping file.

Creating POJO Class With Annotation

RUNNING THE APPLICATION

```
public class Client
{
public static void main(String [] args)
{
Configuration configuration= new Configuration().configure();
StandardServiceRegistry standardRegistry = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
Metadata metaData = new
MetadataSources(standardRegistry).getMetadataBuilder().build(); sessionFactory
= metaData.getSessionFactoryBuilder().build();

}
}
```

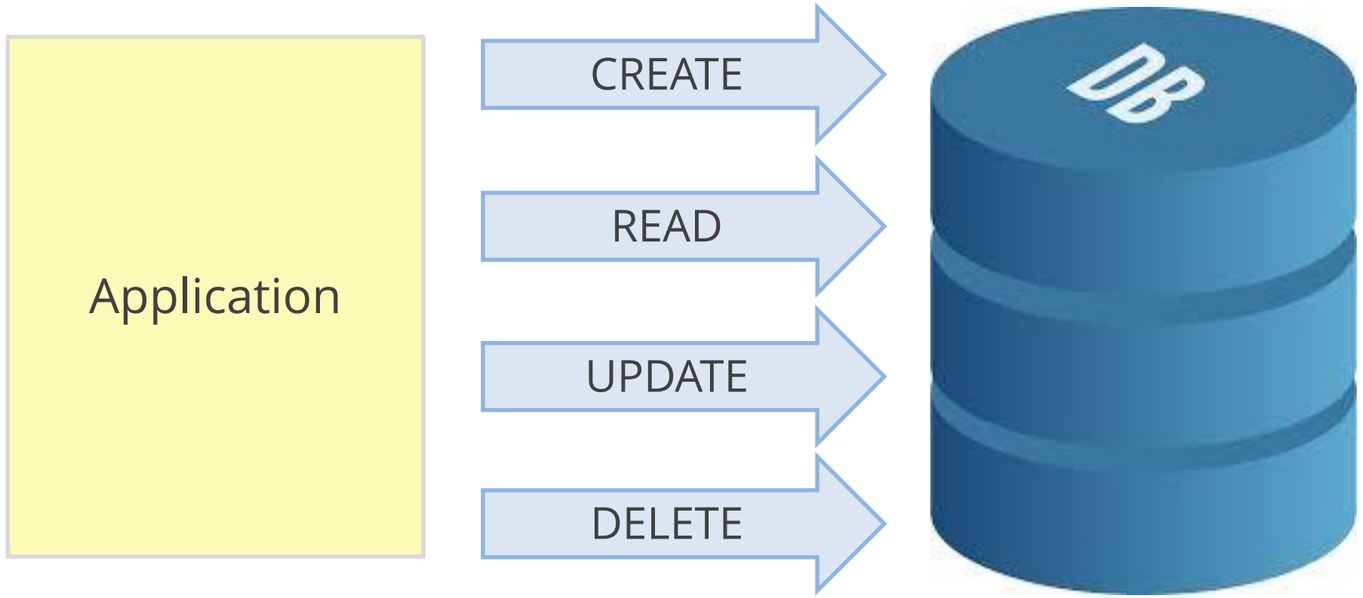
Advanced Java

Topic 6—Hibernate CRUD Operation

CRUD Operation

The acronym CRUD stands for Create, Read, Update, and Delete

They are four basic operations that any data-driven application performs often



Hibernate also supports CRUD operation by means of Session interface.

Hibernate CRUD Operation

LIMIATATIONS OF USING SESSION METHODS

1. Multiple persistent objects can’t be retrieved
2. Multiple delete and update operations are not possible
3. Required criteria can’t be expressed to perform some complex CRUD operation

To overcome the above limitations, the following are used:

a. HQL (Hibernate Query Language)

b. Criteria API

c. Native SQL (Structured Query Language)

HQL

HQL
CRITERIA API
NATIVE SQL

- It gives query language provided by Hibernate.
- It is same as SQL (Structured Query Language), but it doesn't depends on the table of the database. It uses classes.

Criteria

HQL

CRITERIA API

NATIVE SQL

- It is one way of querying the DB in the hibernate application
- It can be used for DML (Database Manipulation Language) operations
- Criteria API is an object oriented after native for HQL to read data from database
- Criteria API supports compile time checking for the query that we build, unlike HQL

Hibernate CRUD Operation

CREATE

Create and insert: The student table is created with two columns, rollNo and name. Let's add data for one student through the following code:

- CREATE
- READ
- UPDATE
- DELETE

```
public class Client
{
public static void main(String [] args)
{
Configuration configuration= new Configuration().configure();
StandardServiceRegistry standardRegistry = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
Metadata metaData = new
MetadataSources(standardRegistry).getMetadataBuilder().build();
sessionFactory = metaData.getSessionFactoryBuilder().build();
Session s = factory.openSession();
Transaction t=s.beginTransaction();
Student s1=new Student ();
s1.setRollNo(1);
s1.setName("yachaan");
s.save(s1);
t.commit();
s.close();
}
}
```

POJO class and hibernate.cfg.xml file remains same

Hibernate CRUD Operation

READ

To read and retrieve data, create a Student object with existing ID and sessionobject.get (Class clazz, Serializable id).

- CREATE
- READ
- UPDATE
- DELETE

```
public class Client
{
public static void main(String [] args)
{
Configuration configuration= new Configuration().configure();
StandardServiceRegistry standardRegistry = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build(); Metadata
metaData = new MetadataSources(standardRegistry).getMetadataBuilder().build();
sessionFactory = metaData.getSessionFactoryBuilder().build();
Session s = factory.openSession();
Transaction t=s.beginTransaction();
// now we are retrieving students record
Student s1=s.get(Student.class,1);
System.out.print(s1.getId()+”  “ + s1.getName());
}
}
```

Hibernate CRUD Operation

UPDATE

Create a Student object with existing ID and use sessionobject.update (Object object)

- CREATE
- READ
- UPDATE
- DELETE

```
public class Client
{
public static void main(String [] args)
{
Configuration configuration= new Configuration().configure();
StandardServiceRegistry standardRegistry = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build(); Metadata
metaData = new MetadataSources(standardRegistry).getMetadataBuilder().build();
sessionFactory = metaData.getSessionFactoryBuilder().build();
Session s = factory.openSession();
Transaction t=s.beginTransaction();
// now we are retrieving students record
Student s1=s.get(Student.class,1);
s.update(s1);
}
}
```

Hibernate CRUD Operation

DELETE

Create a Student object with existing ID and use sessionobject.delete (Object object)

- CREATE
- READ
- UPDATE
- DELETE

```
public class Client
{
public static void main(String [] args)
{
Configuration configuration= new Configuration().configure();
StandardServiceRegistry standardRegistry = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build(); Metadata
metaData = new MetadataSources(standardRegistry).getMetadataBuilder().build();
sessionFactory = metaData.getSessionFactoryBuilder().build();
Session s = factory.openSession();
Transaction t=s.beginTransaction();
// now we are retrieving students record
Student s1=s.get(Student.class,1);
s.delete(s1);
}
}
```

Key Takeaways

- Hibernate converts database-specific queries automatically, based on dialect provided. It uses HQL (hibernate Query Language), which is independent of Database.
- As an Object/Relational Mapping (ORM) framework, Hibernate is concerned with data persistence as it applies to relational databases (via JDBC).
- Hibernate annotations is the newest way to define mappings without the use of XML file. You can use annotations in addition to or as a replacement of XML mapping metadata.
- Hibernate supports CRUD operation by means of Session interface.

Quiz

QUIZ 1

Which of the following is the hibernate configuration file?

a. hibernate.xml

b. hibernate.config.xml

c. hibernate.properties

d. hibernate.cfg.xml

QUIZ 1

Which of the following is the hibernate configuration file?

a. hibernate.xml

b. hibernate.config.xml

c. hibernate.properties

d. hibernate.cfg.xml

The correct answer is **d.**

hibernate.cfg.xml is the hibernate configuration file.

QUIZ 2

Which of the following is true about Hibernate?

a. Hibernate is an open source framework

b. Hibernate is a framework based on business logic

c. Hibernate is an ORM tool

d. Hibernate doesn’t use JDBC internally

QUIZ 2

Which of the following is true about Hibernate?

a. Hibernate is an open source framework

b. Hibernate is a framework based on business logic

c. Hibernate is an ORM tool

d. Hibernate doesn't use JDBC internally

The correct answer is **a and c.**

Hibernate is a open source framework and an ORM tool.

Thank You