**Introduction**

All applications requires some persistent mechanism to store application generated data. In java we can store data using variables and Objects. But these variables and Objects is will be stored into Secondary memory. This secondary memory will be available till the application is executing. Once application execution completed this secondary memory will be vanished, then data which is stored in secondary memory also will get vanished. So we will lose the data if store in variables and Objects. To overcome this problem we need to go for Persistence stores.

**What is Persistence and what is Persistence store?**

The process of storing and maintaining the data for long time is called Persistence. To store and maintain data for long time we need Persistence stores. Below are the Persistence stores available

1. File 2. RDBMS

Files are used to store small amount of data.

Databases are used to store huge amount of data and these are having lot of advantages when compared with the files.

After storing the data into Persistence stores, we can perform some operations on Persistent data, these operations are called as Persistence operations. Below are the persistence operations

CURD or CRUD

C- Create /Insert

U – Update

R – Retrieve D – Delete

To perform these persistence operations on persistence data we need Persistence Technologies.

JDBC is one of the Persistence Technology. It is released by Sun micro system as part of JDK 1.1v.

**Drawbacks of JDBC**

* In JDBC, we have to write lot of boiler plate of code to perform persistence operations
* Here as a programmer every time we need to open and close the connection (Opening and closing connection for multiple times will decrease Performance of the application)
* JDBC supports only Database dependent queries
* In JDBC we need to write SQL queris in various places, after the program has created if the table structure is modified then the JDBC program doesn’t work, again we need to modify and compile and re-deploy required, which is tedious work.
* Every line of JDBC code will throw Checked Exception(java.lang.SQLException), as a programmer we have to handle all these exceptions
* JDBC used to generate database related error codes if an exception occurs, but java programmers are unknown about this error codes.
* There is AUTO DDL support in JDBC and programmer is responsible to generate Primary Keys for the tables.

In order to overcome above problems, Hibernate came into picture..!!

**What is Hibernate?**

[Hibernate](http://www.hibernate.org/) is an open-source light weight ORM framework.

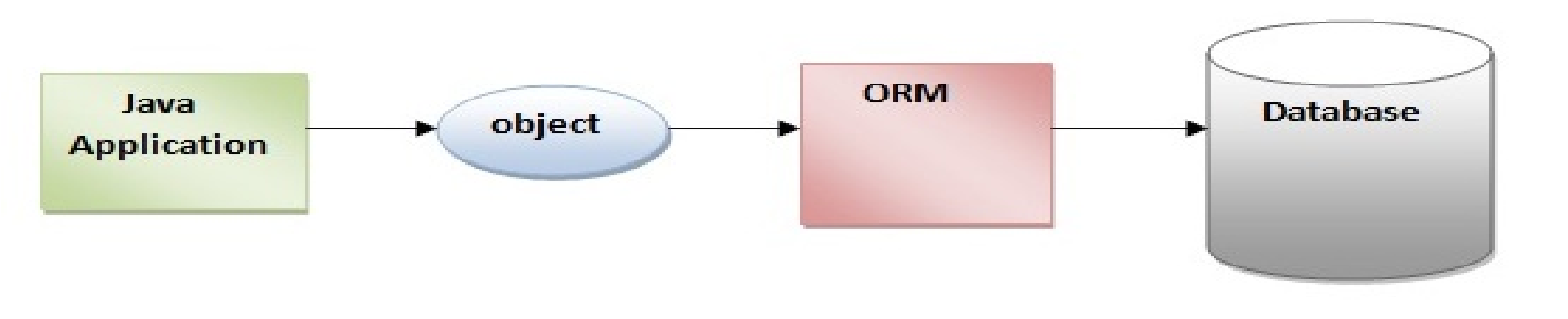
Hibernate was started in 2001 by Gavin King with colleagues from Cirrus Technologies as an alternative to using EJB2-style entity beans. The original goal was to offer better persistence capabilities than those offered by EJB2, by simplifying the complexities and supplementing certain missing features.

In early 2003, the Hibernate development team began Hibernate2 releases, which offered many significant improvements over the first release.

[JBoss, Inc.](https://en.wikipedia.org/wiki/JBoss_(company)) (now part of [Red Hat)](https://en.wikipedia.org/wiki/Red_Hat) later hired the lead Hibernate developers in order to further its development.

Actually Hibernate is much more than ORM Tool (Object - Relational Mapping) because today it is providing lots of features in the persistence data layer.

**ORM means Objects to Relational Mapping i.e. Java Persistence Objects are mapped to Relational databases by using ORM tools.**



**Advantages of Hibernate**

There are many advantages of Hibernate Framework. They are as follows:

*ORM*

Hibernate ORM easily solves the data mismatch found between the object oriented classes of an application and relational database. ORM connects these two with ease through the use of the XML mapping file. It enables to gain complete control over the application as well the database design. This feature makes Hibernate flexible and powerful.

*Transparent Persistence*

Hibernate’s transparent persistence ensures automatic connection between the application’s objects with the database tables. This feature prevents developers from writing lines of connection code. Transparent persistence enables hibernate to reduce the development time and maintenance cost.

*Database independent*

Hibernate is database independent. It can be used to connect with any database like Oracle, MySQL, Sybase and DB2 to name a few. This cross database portability of Hibernate is easily achieved by changing a parameter ‘database dialect’ in the configuration file. Database independency is considered as one of the major advantages of Hibernate.

*HQL*

Hibernate supports a powerful query language called HQL (Hibernate Query Language). This query language is more powerful than SQL and is completely object oriented. HQL’s advanced features like pagination and dynamic profiling are not present in SQL.

HQL can be used to implement some of the prominent object oriented concepts like inheritance, polymorphism and association.

*Dual-layer Caching*

Hibernate supports both first level and second level caching mechanisms. The first level caching is associated with Session object which is used by default. The second level caching is associated with Session Factory object.

Through caching concept, Hibernate retains the objects in cache so as to reduce repeated hits to the database. This feature makes Hibernate highly scalable and optimizes the application’s performance.

*Version Property*

Hibernate supports optimistic locking through its version property feature. This functionality supports multiple transactions without affecting one another.

For example, when two or more users try to alter a database entity at the same time, the version field avoids the conflict and gives preference to the user who commits the changes first. The other user will be prompted with an error message and will be asked to restart the process.

*Open Source*

Hibernate is available as an open source software with zero cost product license. This light weight software can be downloaded from its source website [hibernate.org.](http://hibernate.org/)

*Scalability*

Hibernate is highly scalable. It adapts itself in any environment. It may be an intranet application with few hundreds of users or large critical application with thousands of users. Hibernate supports both the applications equally.

*Lazy-Loading*

The lazy-loading concept fetches only the necessary object that is required for the execution of an application.

For example, if there is one parent class and n number of child classes, during an execution, there is no need to load all the child classes. Instead, only the class that is required for the query or join need to be loaded. This concept of lazy-loading prevents the unnecessary loading of objects. It enhances the performance of the application.

**Why We Have to Use ORM Tools?**

Object-relational mapping, in the purest sense, is a programming technique that supports the conversion of incompatible types in object-oriented programming languages, specifically between a data store and programming objects. You can use an ORM framework to persist model objects to a relational database and retrieve them, and the ORM framework will take care of converting the data between the two otherwise incompatible states. Most ORM tools rely heavily on metadata about both the database and objects, so that the objects need to know nothing about the database and the database doesn’t need to know anything about how the data is structured in the application. ORM provides a clean separation of concerns in a well-designed data application, and the database and application can each work with data in its native form.

The key feature of ORM is the mapping it uses to bind an object to its data in the database. Mapping expresses how an object and its properties and behaviors are related to one or more tables and their fields in the database. An ORM uses this mapping information to manage the process of converting data between its database and object forms, and generating the SQL for a relational database to insert, update, and delete data in response to changes the application makes to data objects.

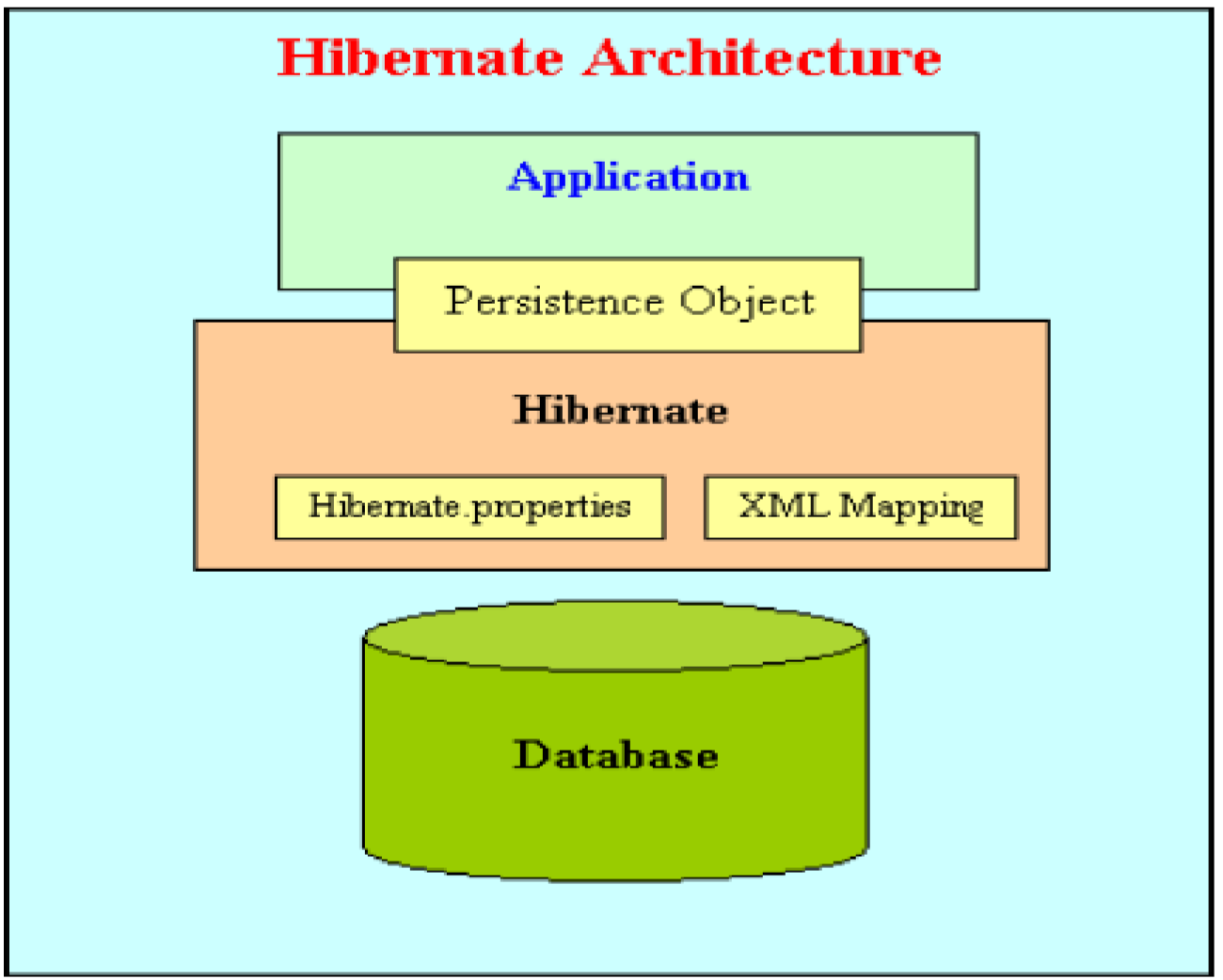
There are a number of benefits to using an ORM for development of databased applications

* **Productivity:** The data access code is usually a significant portion of a typical application, and the time needed to write that code can be a significant portion of the overall development schedule. When using an ORM tool, the amount of code is unlikely to be reduced—in fact, it might even go up—but the ORM tool generates 100% of the data access code automatically based on the data model you define, in mere moments.
* **Application design**: A good ORM tool designed by very experienced software architects will implement effective design patterns that almost force you to use good programming practices in an application. This can help support a clean separation of concerns and independent development that allows parallel, simultaneous development of application layers.
* **Code Reuse**: If you create a class library to generate a separate DLL for the ORM-generated data access code, you can easily reuse the data objects in a variety of applications. This way, each of the applications that use the class library need have no data access code at all.
* **Application Maintainability**: All of the code generated by the ORM is presumably well tested, so you usually don’t need to worry about testing it extensively. Obviously you need to make sure that the code does what
* you need, but a widely used ORM is likely to have code banged on by many developers at all skill levels. Over the long term, you can refactor the database schema or the model definition without affecting how the application uses the data objects.

## Hibernate Architecture

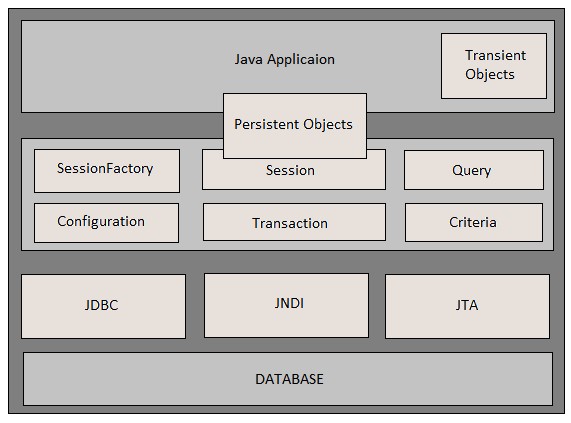
The Hibernate architecture is layered to keep us isolated from having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.

There are 4 layers in hibernate architecture they are, java application layer, hibernate framework layer, backhand api layer and database layer. Let's see the diagram of hibernate architecture:



Java program will expose the data in the form of objects, Using Hibernate we can store those objects into the database directly.

**Internal Structure of Hibernate application**



# Configuration

Configuration object is created to load the configuration files like hibernatate.cfg.xml and hbm files (Java objects to database mapping). Since this step is loading of configurations, it generally happens at the application initialization time.

Configuration cfg = new Configuration();

# SessionFactory

We would always need one instance of Session factory per database that our application is interacting with. So if we have two different databases, we would create two session factory objects.

As Session factory is a heavy weight object, creation of the session factory object is an expensive operation and recommended to get it created at application start up.

Obtaining sessionFactory

====================================================================== **protected void setUp() throws Exception {**

**// A SessionFactory is set up once for an application!**

**final StandardServiceRegistry registry = new StandardServiceRegistryBuilder() .configure() // configures settings from hibernate.cfg.xml**

**.build();**

**try{**

**sessionFactory = new MetadataSources( registry**

**).buildMetadata().buildSessionFactory();**

**}**

**catch (Exception e) {**

**// The registry would be destroyed by the SessionFactory, but we had trouble building the SessionFactory**

**// so destroy it manually.**

**StandardServiceRegistryBuilder.destroy( registry );**

**}**

**}**

The setUp method first builds a org.hibernate.boot.registry.StandardServiceRegistry instance which incorporates configuration information into a working set of Services for use by the SessionFactory. In this tutorial we defined all configuration information in hibernate.cfg.xml so there is not much interesting to see here.

Using the StandardServiceRegistry we create the org.hibernate.boot.MetadataSources which is the start point for telling Hibernate about your domain model. Again, since we defined that in hibernate.cfg.xml so there is not much interesting to see here.

org.hibernate.boot.Metadata represents the complete, partially validated view of the application domain model which the SessionFactory will be based on.

The final step in the bootstrap process is to build the SessionFactory. The SessionFactory is a thread-safe object that is instantiated once to serve the entire application.

The SessionFactory acts as a factory for org.hibernate.Session instances, which should be thought of as a corollary to a "unit of work".

# Session

Session objects are created using Session factory and are a lightweight object. Session objects provide a connection with a relational database.

By design, we should create a new session object when a database interaction is needed. A session object represents a unit of work.

Session objects are not thread safe and hence should not keep open for a long time.

|  |
| --- |
| Session session = sessionFactory.openSession(); |

***Transaction***

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

Transaction tx = session.beginTransaction();//start

//operations

tx.commit(); //end

## Hibernate Application Requirements

Any hibernate application, for example consider even first hello world program must always contains 4 files totally.

* POJO class
* Mapping XML or Annotations
* Configuration XML
* One java file to write our business logic

Actually these are the minimum requirement to run any hibernate application, and in fact we may require any number of POJO classes and any number of mapping xml files (Number of POJO classes = that many number of mapping xmls), and only one configuration xml and finally one java file to write our logic.

**Hibernate Persistent Classes or POJO classes**

Persistent classes are those java classes whose objects have to be stored in the database tables. They should follow some simple rules of Plain Old Java Object programming model (POJO).

1. A persistence class should have a default constructor
2. A persistence class should have an id to uniquely identify the class objects.All attributes should be declared as private
3. Public getter and setters shold be defined to access the class attribues

*===================Product.java==========================* **public class Product {**

**private Integer pid;**

**private String pname;**

**private Double price;**

**public Integer getPid() {**

**return pid;**

**}**

**public void setPid(Integer pid) {**

**this.pid = pid;**

**}**

**public String getPname() { return pname;**

**}**

**public void setPname(String pname) { this.pname = pname;**

**}**

**public Double getPrice() { return price;**

**}**

**public void setPrice(Double price) { this.price = price;**

**}**

**}**

==========================================================================

**Hibernate configuration file**

Hibernate works as an intermediate layer between java application and relational database. So hibernate needs some configuration setting related to the database and other parameters like mapping files.

*A hibernate configuration file mainly contains three types of information*

* Connection Properties related to the database.
* Hibernate Properties related to hibernate behavior.
* Mapping files entries related to the mapping of a POJO class and a database table.
* *Note: No. of hibernate configuration files in an application is dependence upon the no. of database uses. No. of hibernate configuration files are equals to the no. of database uses.*

<hibernate-configuration>

<session-factory>

// Connection Properties

<property name=*"connection.driver\_class"*>driverClassName</property>

<property name=*"connection.url"*>jdbcConnectionURL</property>

<property name=*"connection.user"*>databaseUsername</property>

<property name=*"connection.password"*>databasePassword</property>

// Hibernate Properties

<property name=*"show\_sql"*>true/false</property>

<property name=*"dialet"*>databaseDialetClass</property>

<property name=*"hbm2ddl.auto"*>like create/update</property>

// Mapping files entries

<mapping resource=*"mappingFile1.xml"* />

<mapping resource=*"mappingFile2.xml"* />

</session-factory>

</hibernate-configuration>

## Hibernate mapping file

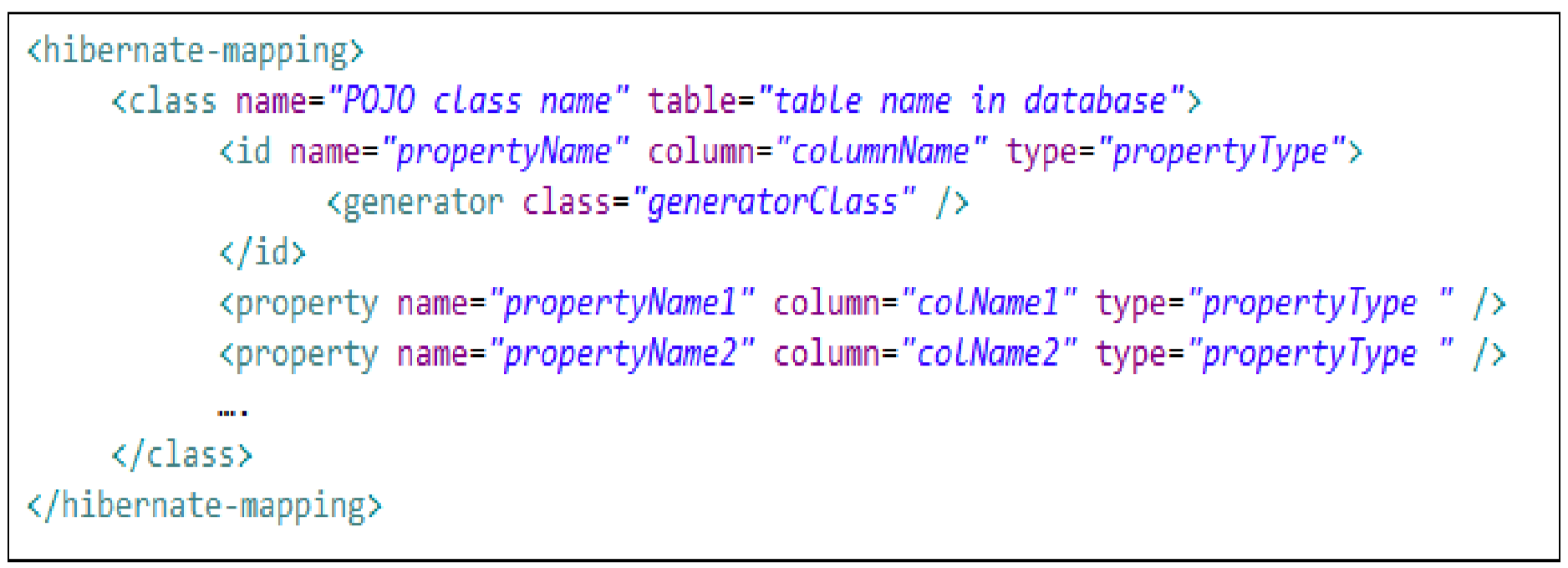
Hibernate mapping file is used by hibernate framework to get the information about the mapping of a POJO class and a database table.

*It mainly contains the following mapping information*

* Mapping information of a POJO class name to a database table name.
* Mapping information of POJO class properties to database table columns.

*Elements of the Hibernate mapping file:*

* 1. **hibernate-mapping:** It is the root element.
  2. **Class:** It defines the mapping of a POJO class to a database table.
  3. **Id:** It defines the unique key attribute or primary key of the table.
  4. **generator:** It is the sub element of the id element. It is used to automatically generate the id.
  5. **property:** It is used to define the mapping of a POJO class property to database table column.



## Hibernate First Application

To create Hibernate application we need to create below files

Product.java (POJO class)

Product.hbm.xml (Xml mapping file ) hibernate.cfg.xml (Xml configuration file)

ClientForSave.java (java file to write our hibernate logic)

==========================Product.java================================= **public class Product {**

**private Integer productId; private String proName;**

**private Double price;**

**//Setters and getters**

**}**

=========================================================================

**<?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=*"Product"* table=*"PRODUCTS"*>**

**<id name=*"productId"* type=*"int"* column=*"PID"*>**

**</id>**

**<property name=*"procName"* column=*"PRODUCT\_NAME"* />**

**<property name=*"price"* column=*"PRICE"* />**

**</class>**

**</hibernate-mapping>**

=====================-Product.hbm.xml==================================

In this mapping file, Product class is linked with PRODUCTS table in the database, and next is the id element, means in the database table what column we need to take as primary key column, that property name we need to give here, here property name “productId” which will mapped with “pid “ column in the table.

And “proName” is mapped with “pname” column of the PRODUCTS table, here we have not specified any column for the property price, this means that, our property name in the pojo class and the column name in the table both are same.

Remember: the first 3 lines is the DTD for the mapping file, as a programmer no need to remember but we need to be very careful while we are copying this DTD, program may not be executed if we write

DTD wrong, actually we have separate DTD’s for Mapping xml and Configuration xml files. ======================**hibernate.cfg.xml**====================================

**<?xml version=*"1.0"* encoding=*"utf-8"*?>**

**<!DOCTYPE hibernate-configuration SYSTEM**

**"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">**

**<hibernate-configuration>**

**<session-factory>**

**<!-- Related to Database information -->**

**<property name=*"hibernate.connection.driver\_class"*> oracle.jdbc.driver.OracleDriver**

**</property>**

**<property name=*"hibernate.connection.url"*> oracle:jdbc:thin:@localhost:1521/XE**

**</property>**

**<property name=*"hibernate.connection.username"*>**

**hibernate**

**</property>**

**<property name=*"hibernate.connection.password"*>**

**hibernate@123**

**</property>**

**<!-- Related to Hibernate Properties -->**

**<property name=*"show\_sql"*>true</property>**

**<property name=*"hbm2ddl.auto"*>update</property>**

**<property**

**name=*"dialect"*>org.hibernate.dialect.OracleDialect</property>**

**<!-- List of XML mapping files -->**

**<mapping resource=*"Prodcut.hbm.xml"* />**

**</session-factory>**

**</hibernate-configuration>** ========================SavePRoduct.java============================= **public class SaveProduct {**

**public static void main(String[] args) { Configuration cfg = new Configuration() cfg.configure("hibernate.cfg.xml"); StandardServiceRegistry registry = new**

**StandardServiceRegistryBuilder().configure().build();**

**SessionFactory sessionFactory = new MetadataSources( registry**

**).buildMetadata().buildSessionFactory();**

**Session hsession = sessionFactory.openSession(); Transaction tx = hsession.beginTransaction();**

**Product p = new Product(); p.setProductId(101);**

**p.setProName("Apple");**

**p.setPrice(35000.50);**

**hsession.save(p); tx.commit(); hsession.close(); sessionFactory.close();**

**}**

**}**

=========================SaveProduct.java========================

Example1:

----------------

We need to download hibernate-core jar dependency and configure in pom.xml

1)Create a Project named as demoOnHibernate

2)Congfiure hibernate,mysql connecter dependeny in pom.xml

3) Create a object relation mapping (orm) between ProductPojo class and product table(product.hbm.xml)

4)create a configuration file which contains properties to interact with mysql

(driverclass,url,username,password)

5) create a java class to perform insert into product tables using session Object.

Example 2:

---------------

We need to download hibernate-core5.4.33 jar dependency and configure in pom.xml

1) Create a Project named as demoOnHibernteWithAnnotation

2) Configure hibernate(5.4.33),mysql connecter dependency in pom.xml

3) Create a object relation mapping (orm) between Course Pojo class and course table using annotation

4) Create a configuration file which contains properties to interact with mysql(driverclass,url,username,password)

5) Create a java class to perform insert into Course table tables using session Object.

Example 3:

We need to download hibernate-core5.4.33 jar dependency and configure in pom.xml

1) Create a Project named as demoOnHibernteWithAnnotationUseCase1

2) Configure hibernate(5.4.33),mysql connecter dependency in pom.xml

3) Create a object relation mapping (orm) between Employee(int empId,empName,empSal) Pojo class and Emp(empno,ename(20),sal(10,2))

table using annotations(javax.persistance)

Note : create the table emp table from hibernate

4) Create a configuration file which contains properties to interact with mysql(driverclass,url,username,password)

5) Create a java class to perform insert 5 records into Employee Objects (records)table tables using session Object.

6) Create a java class to perform delete a record based on id Employee table tables using session Object.

7able tables using session Object.

) Create a java class to perform update a record based on id Employee t 8) Create a java class to perform search a record based on id from Employee table tables using session Object.

9) Create a java class to list all records from Employee table tables using session Object.

Methods:

1. save()
2. delete()
3. update()
4. load()
5. loadAll()
6. saveOrUpdate()

save the file with id\_name\_hibernate.doc

please upload this above assignment on sprringlab url

https://docs.google.com/forms/d/e/1FAIpQLSdoBEWLACvHZ5I6VrRg\_lo-zW3CgLkOKJ2I1NIu4MBy2WC2xQ/viewform