**Hibernate Basics Note**

Solution - Object Relational Mapping

Object-Relational Impedance Mismatch is one of the key challenge in data persistence and can be resolved using **Object Relational Mapping**.

We use **ORM** mainly for following reasons:

* ORM resolves the Object-Relational Paradigm mismatch.
* The lower level interaction with database is handled by the ORM. Framing and executing the database dependent queries is taken care by the ORM framework.
* ORM helps the developer to get rid of “messy SQL”. The developer need not waste time in writing the plumbing code.
* ORM allows the developer to concentrate on the business logic and work with the object model.
* ORM is database independent. All database vendors provide support for ORM. Hence the application code becomes portable without worrying about the underlying database.

Because of the challenge of importing applications from one ORM framework to another **Java Persistent API** was developed.

**Java Persistence API (JPA)** was released by Java Community, to standardize the persistence process. JPA incorporated many features from the existing frameworks like Hibernate and TopLink Essentials and then it became the standard specification for ORM in Java.

**ORM**

Object Relational Mapping (ORM) is a programming technique for mapping Java Entity class to a relational database table, where

* **Java Entity class** is mapped to a **table**
* **Data members** of a class are mapped to **table columns**
* **Instance** of a Java Entity class is mapped to a**record**in a table

Hibernate is an open source framework which provides implementation for JPA Specification and it is flexible and powerful ORM solution to map Java classes to database tables.

**Features of Hibernate:**

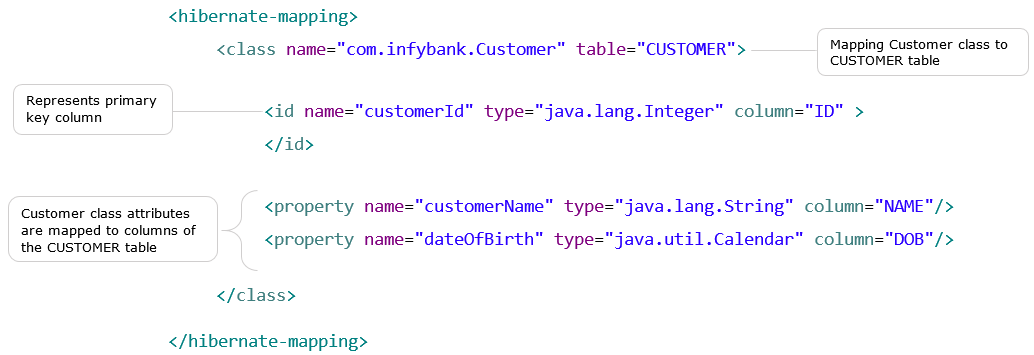
* **Object Relational Mapping:** Hibernate, being a ORM framework aims to resolve the Object-Relational Impedance Mismatches proving itself as an effective data persistence medium.
* **Scalability and Reliability:** Hibernate works well in client-server based environment and delivers a scalable architecture. Hibernate provides good stability and quality, hence it is reliable.
* **Extensible:** Hibernate is highly configurable and extensible.
* **High Performance:**Hibernate has high performance due to various features like multiple fetch strategies, optimistic locking with automatic versioning and time stamping, caching etc.
* **Idiomatic** **Persistence:**Hibernate enables development of persistent classes which follows object-oriented idioms like inheritance, polymorphism, association, composition and the Java collections framework.

**Benefits of Hibernate:**

* **Lightweight:**Hibernate implements ORM using simple POJO classes.
* **Open Source:** Hibernate is freely available and may be redistributed and modified
* **Vendor Independent:** Hibernate, or in general JPA, prevents writing code according to the database vendor. Hence it is vendor independent.
* **Non Invasive:**Hibernate does not force the developer to extend or implement any class or interface.

Hibernate Mapping Using XML

To map entity class **com.infybank.Customer** to **Customer**table, the mapping file **Customer.hbm.xml** is created as shown below:

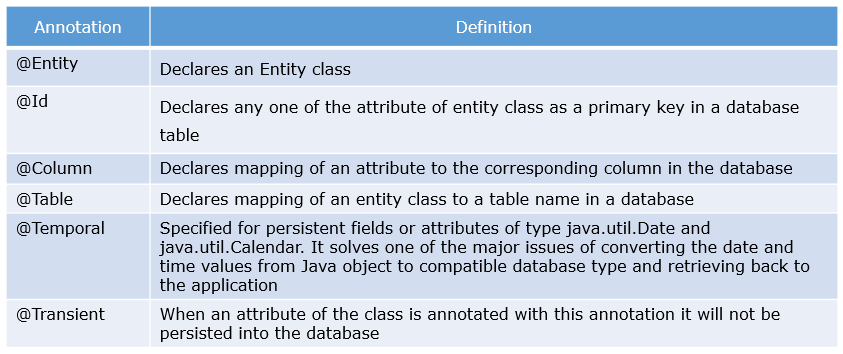


Hibernate data types are available to map data types supported in Java and databases.

Hibernate Mapping Using Annotations

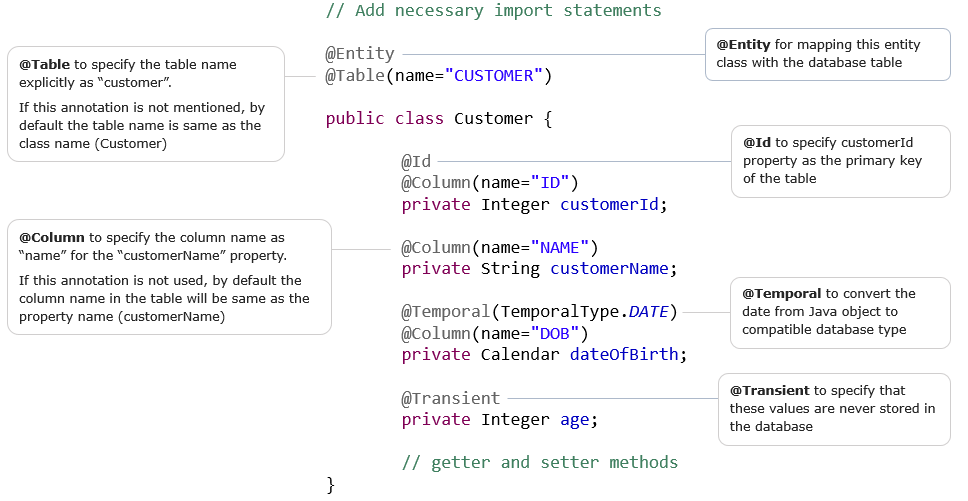
The mapping information can also be provided through**annotations** instead of defining ***\*.hbm.xml*** file. Annotations can be directly given in the entity classes.

Few of the common annotations used are available in **javax.persistence**package:



### **Hibernate Mapping Using Annotations**

In the previous Infy Bank Customer example let us use annotations for hibernate mapping instead of .hbm file.



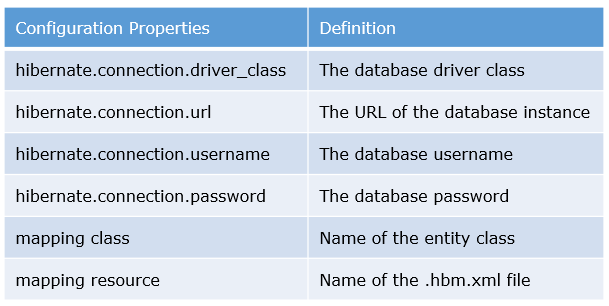
Hibernate Configuration

Hibernate requires to know in advance, where to find the mapping information between the Java classes and the related database tables. Hibernate also needs to know the information related to database and other related parameters. This can be provided using **Hibernate Configuration**.

Hibernate Configuration has the following information.

* Database connection settings
* Hibernate properties
* Mapping Resource details

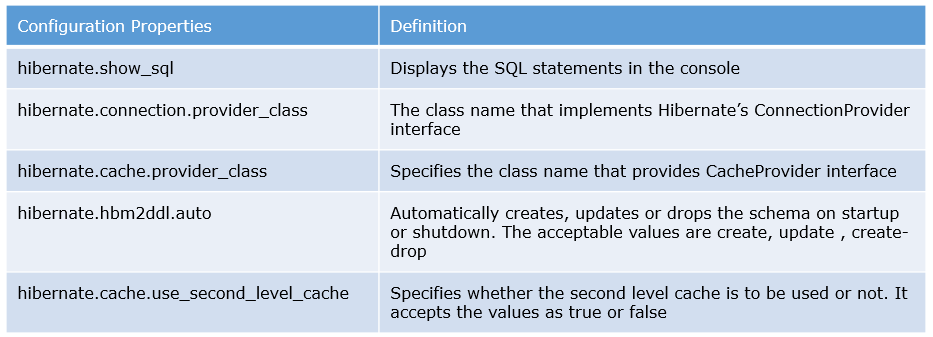
Following properties are defined in the Hibernate Configuration:



**Note**: For hibernate configuration either mapping class or mapping resource needs to be used.

Hibernate Configuration

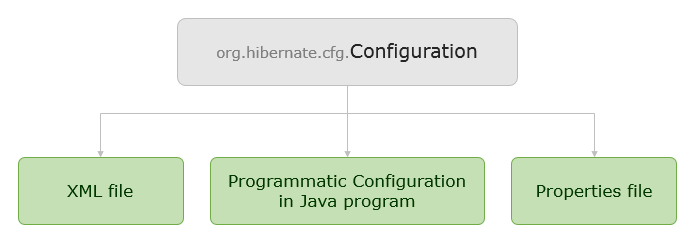
Few additional, optional Hibernate properties that can be defined in the configuration files are:



Hibernate Configuration

The configuration details can be provided using any of the below approaches:

* **XML file** **based Configuration**: A xml file, usually named as**hibernate.cfg.xml**, is defined  with all the required properties, and is placed in the classpath of the application.
* **Programmatic Configuration**: An instance of  org.hibernate.cfg.**Configuration**  class is created and set with the required properties in the .java file.
* **Property file based Configuration**: A **.properties** file is defined with the required configuration properties.



Hibernate Configuration - XML File Based Configuration

The Infy Bank Application is connecting to MySQL database(hibernatedb), the details of the database connection and the mapping details of customer entity is provided using hibernate.cfg.xml as shown below.

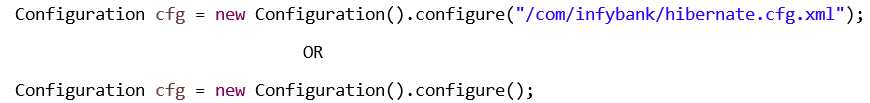


Hibernate Configuration - XML File Based Configuration

To start using the configuration file, our Java application must create a **Configuration** instance first. This is available in **org.hibernate.cfg**package**.**

The properties and mapping configured in xml file or property file is read and loaded by Hibernate, into the Configuration instance created. The Configuration is meant only as an initialization-time object. The **Configuration** instance can be created as below :

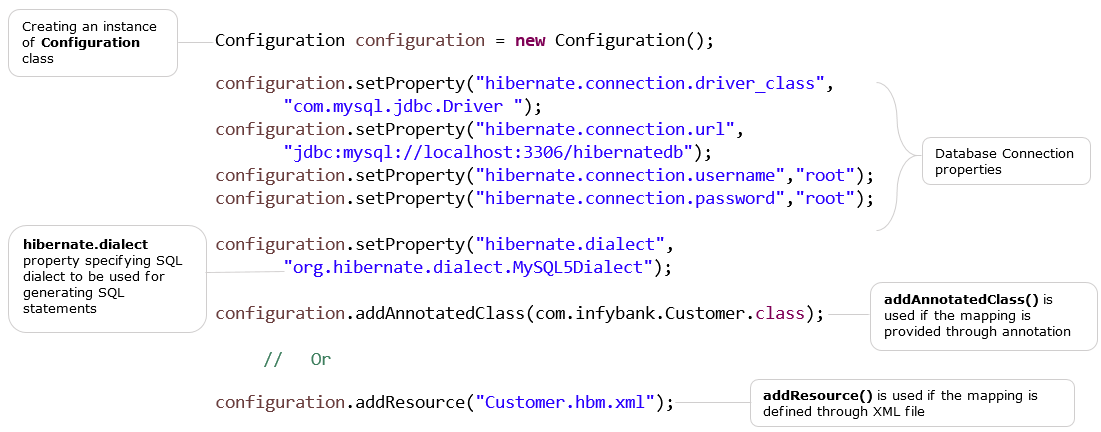
   Assuming the configuration file resides in the package**”com.infybank”**.



**Note:**If the configuration file is in classpath with the name as “**hibernate.cfg.xml**”, then Hibernate  automatically loads the configuration file. There is no need to mention the file name as an argument to **configure()** method.

Hibernate Configuration - Programmatic Configuration

Instead of using hibernate.cfg.xml, the hibernate configuration can be provided programatically using Configuration class's **setProperty()** method which is demonstrated below:



It is not required to invoke the **Configuration.configure()** method in this example, as the properties are directly loaded in the Configuration instance.

Hibernate Configuration - Properties Configuration

Hibernate configuration can be configured in **hibernate.properties** file. This file has to be placed in root directory of application’s classpath. A sample hibernate.properties file and loading of these properties in Configuration instance is as shown below:

