**Draw Backs of JDBC:**

* In JDBC, if we open a database connection we need to write in try, and if any exceptions occurred catch block will takers about it, and finally used to close the connections.
* here as a programmer we must close the connection, or we may get a chance to get our of connections message…!
* Actually if we didn’t close the connection in the finally block, then jdbc doesn’t responsible to close that connection.
* In JDBC we need to write Sql commands 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.
* JDBC used to generate database related error codes if an exception will occurs, but java programmers are unknown about this error codes right.
* In the Enterprise applications, the data flow with in an application from class to class will be in the form of objects, but while storing data finally in a database using JDBC then that object will be converted into text.  Because JDBC doesn’t transfer objects directly.

# What is Hibernate:

Hibernate is the ORM tool given to transfer the data between a java (object) application and a database (Relational) in the form of the objects.

Hibernate is a non-invasive framework,  means it wont forces the programmers to extend/implement any class/interface, and in hibernate we have all POJO classes so its light weight.’ Hibernate can runs with in or with out server, i mean it will suitable for all types of java applications (stand alone or desktop or any servlets bla bla.)

Hibernate is purely for persistence (to store/retrieve data from Database).

Mapping and Configuration are very familiar keywords we used to here in the hibernate, every hibernate program must need these 2 xml files.

**Mapping:**

* Mapping file is the heart of hibernate application.
* Every ORM tool needs this mapping, mapping is the mechanism of placing an object properties into column’s of a table.
* Mapping can be given to an ORM tool either in the form of an XML or in the form of the annotations.
* The mapping file contains mapping from a pojo class name to a table name and pojo class variable names to table column names.
* While writing an hibernate application, we can construct one or more mapping files, mean a hibernate application can contain any number of  mapping files.

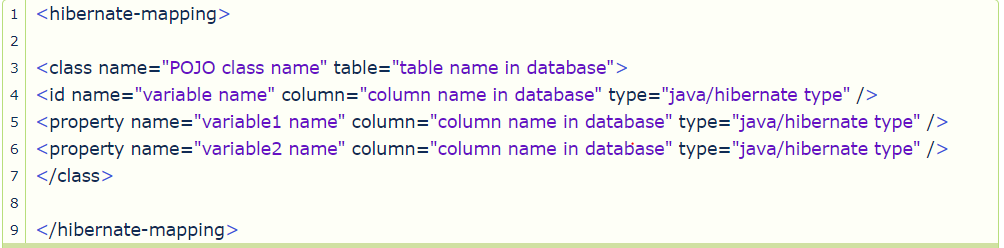
generally an object contains 3 properties like

* Identity (Object Name)
* State (Object values)
* Behavior (Object Methods)

But while storing an object into the database, we need to store only the values(State) right ? but how to avoid identity, behavior.. its not possible. In order to inform what value of an object has to be stored in what column of the table, will be taking care by the mapping,  actually mapping can be done using 2 ways,

* XML
* Annotations.

Actually annotations are introduced into java from JDK 1.5.



**Configuration:**

Configuration is the file loaded into an hibernate application when working with hibernate, this configuration file contains 3 types of information..

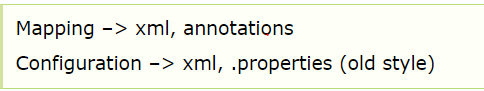
* Connection Properties
* Hibernate Properties
* Mapping file name(s)

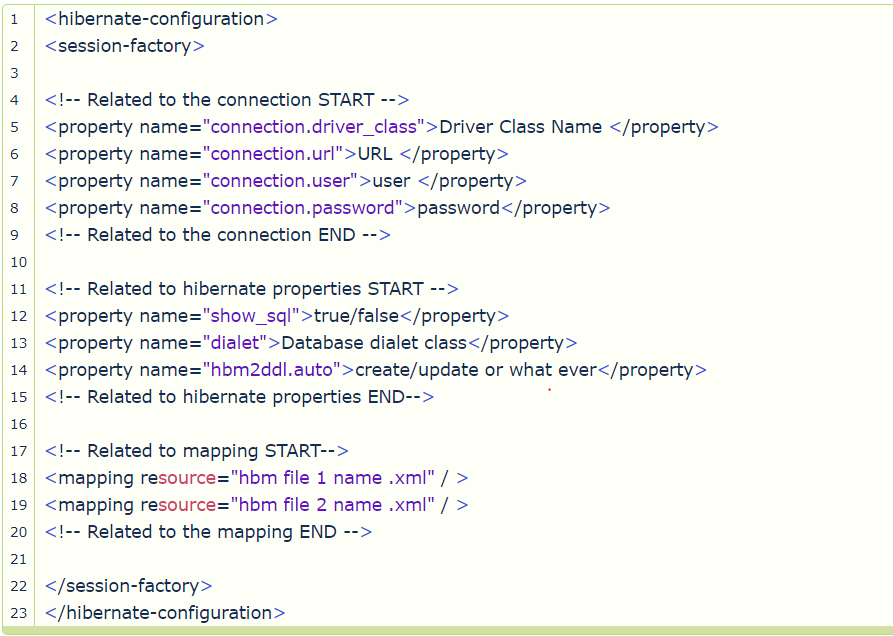
We must create one configuration file for each database we are going to use, suppose if we want to connect with 2 databases, like Oracle, MySql, then we must create 2 configuration files.

No. of databases we are using  = That many number of configuration files

We can write this configuration in 2 ways…

* xml
* By writing Properties file.  We don’t have annotations here, actually in hibernate 1, 2.x we defined this configuration file by writing .properties file, but from 3.x xml came into picture.





**Advantages of hibernates:**

* Hibernate supports Inheritance, Associations, Collections
* In hibernate if we save the derived class object,  then its base class object will also be stored into the database, it means hibernate supporting inheritance
* Hibernate supports relationships like One-To-Many,One-To-One, Many-To-Many-to-Many, Many-To-One
* This will also supports collections like List,Set,Map (Only new collections)
* In jdbc all exceptions are checked exceptions, so we must write code in try, catch and throws, but in hibernate we only have Un-checked exceptions, so no need to write try, catch, or no need to write throws.  Actually in hibernate we have the translator which converts checked to Un-checked
* Hibernate has capability to generate primary keys automatically while we are storing the records into database
* Hibernate has its own query language, i.e hibernate query language which is database independent
* So if we change the database, then also our application will works as HQL is database independent
* HQL contains database independent commands
* While we are inserting any record, if we don’t have any particular table in the database, JDBC will rises an error like “View not exist”, and throws exception, but in case of hibernate, if it not found any table in the database this will create the table for us
* Hibernate supports caching mechanism by this, the number of round trips between an application and the database will be reduced, by using this caching technique an application performance will be increased automatically.
* Hibernate supports annotations, apart from XML
* Hibernate provided Dialect classes, so we no need to write sql queries in hibernate, instead we use the methods provided by that API.
* Getting pagination in hibernate is quite simple.

**Disadvantages of hibernates:**

* I don’t think there are disadvantages in hibernate
* You know some thing.., Its saying hibernate is little slower than pure JDBC, actually the reason being hibernate used to generate many SQL statements in run time, but i guess this is not the disadvantage
* But there is one major disadvantage, which was boilerplate code issue, actually we need to write same code in several files in the same application, but spring eliminated this

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

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

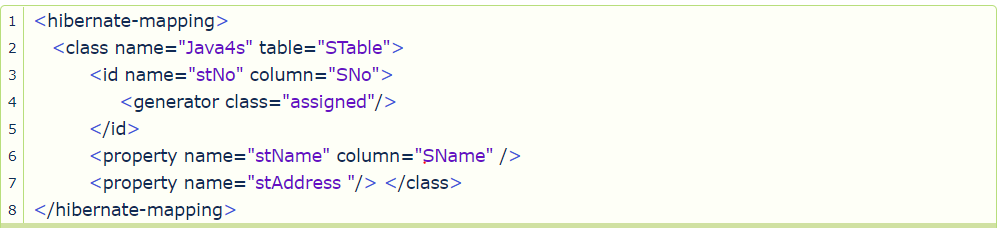
Actually these are the minimum requirement to run any hibernate application, and in fact we may require any number of POJO classes and any number of mapping xml files (**Number of POJO classes = that many number of mapping xmls**), and only one configuration xml and finally one java file to write our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

**POJO Class:**

* POJO is a simple java file, no need to extend any class or implement any interface.
* This POJO class contain private properties variables, and for each property a setter and a getter



## ****Mapping xml For POJO****



Yes., see in this above mapping xml, for stAddress property i have not written any column name i just been specified  **<property name=”stAddress “/>,**this means in the database the column name for stAddress property will also be stAddress, in these cases we can ignore the column attribute to write, and i will explain about this <generator /> element later.

## Configuration XML

Refer above

Usually configuration file name will be hibernate.cfg.xml

# How To Install Hibernate

Download hibernate.jar and use

For maven :

<dependency>

<groupId>org.hibernate</groupId>

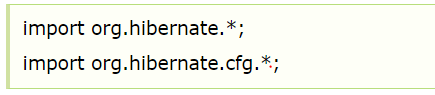
<artifactId>hibernate-core</artifactId>

<version>5.4.0.Final</version>

</dependency>

## Follow The Steps:

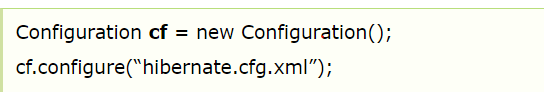
**1.** Import the hibernate API, they are many more, but these 2 are more than enough…



**2.** Among Configuration, Mapping xml files, first we need to load configuration xml, because once we load the configuration file, automatically mapping file will be loaded as we registered this mapping xml in the configuration file.

So to load configuration xml, we need to create object of **Configuration** class, which is given in **org.hibernate.cfg.\***;  and we need to call **configure()** method in that class, by passing xml configuration file name as parameter.

Eg:



Here our configuration file name is your choice, but by default am have been given hibernate.cfg.xml,  so once this configuration file is loaded in our java app, then we can say that hibernate environment is started in our program.

So once we write the line\_ **cf.configure(“hibernate.cfg.xml”),**configuration object**cf**will reads this xml file**hibernate.cfg.xml,**actually internally cf will uses DOM parsers to read the file.

Finally…

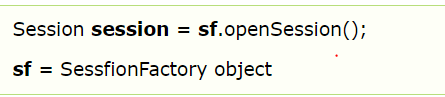
* cf will reads data from hibernate.cfg.xml
* Stores the data in different variables
* And finally all these variables are grouped and create one high level hibernate object we can call as SessionFactory object.
* So Configuration class only can create this SessionFactory object



Actually SessionFactory is an interface not a class, and SessionFactoryImpl is the implimented class for SessionFactory, so we are internally creating object of SessionFactoryImpl class and storing in the interface reference, so this SessionFactory object **sf**contains all the data regarding the configuation file so we can call**sf**as heavy weight object.

3. Creating an object of session,

* Session is an interface and SessionImpl is implemented class, both are given in org.hibernate.\*;
* When ever session is opened then internally a database connection will be opened, in order to get a session or open a session we need to call openSession() method in SessionFactory, it means SessionFactory produces sessions.



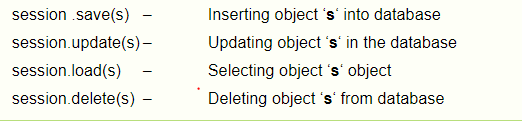
4. Create a logical transaction

While working with insert, update, delete, operations from an hibernate application onto the database then hibernate needs a logical Transaction, if we are selecting an object from the database then we do not require any logical transaction in hibernate.  In order to begin a logical transaction in hibernate then we need to call a method beginTransaction() given by Session Interface.

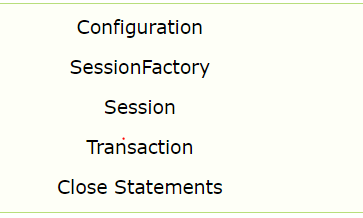
Transaction tx = **sessio**n.beginTransaction();

**session** is the object of Session Interface

5. Use the methods given by Session Interface,  to move the objects from application to database and  from database to application



* So finally we need to call **commit()**in Transaction, like **tx.commit()**;
* As i told earlier,  when we open session a connection to the database will be created right, so we must close that connection as session. close().
* And finally close the SessionFactory as **sf.close()**



Select , delete ,insert code available in project.

**Update**

this is the program to update an object (1 complete row) in the database, which is already persisted in the database, then we have the following two approaches

## Approach 1

Load that object from the database, and modify its values, now hibernate automatically modifies the values on to database also, when ever the transaction is committed.

* When ever an object is loaded from the database then hibernate stores the loaded object in cache-memory maintained by session-interface
* Once an object is loaded, if we do any modifications on that object by calling its setter methods, then these modification are stored in the object maintained by cache-memory
* if we modify the loaded object for multiple times then also the modifications will be stored in object maintained by the cache-memory only.
* when ever we issue commit() operation then hibernate verify whether any changes are there between the object stored in the cache and object in the database, if changes exists then hibernate automatically updates the database by generating any update operation.
* What am saying is hibernate automatically maintains synchronization between cache-memory object and database table objects (rows)

## Approach 2:

If we want to modify object in the database, then create new object with same id and we must call update() given by session interface.

first approach is  recommended always.

# Hibernate Versioning

once an object is saved in a database, we can modify that object any number of times right, If we want to know how many no of times that an object is modified then we need to apply this versioning concept.

When ever we use versioning then hibernate inserts version number as **zero**, when ever object is saved for the first time in the database.  Later hibernate increments that version no by one automatically when ever a modification is done on that particular object.  
In order to use this versioning concept, we need the following two changes in our application

* Add one property of type int in our pojo class
* In hibernate mapping file, add an element called version soon after id element

**Note**:

* Remember friends, first we must run the logic to **save** the object then hibernate will inset 0 (Zero) by default in the version column of the database, its very important point in the interview point of view also
* First save logic to let the hibernate to insert zero in the version column, then any number of update logic’s (programs) we run, hibernate will increments +1 to the previous value
* But if we run the update logic for the first time, hibernate will not insert zero..! it will try to increment the previous value which is NULL in the database so we will get the exception.

Guys your know some thing.., actually we can run any logic (Save or Update) for the first time, but make sure the versioning column is a number (>=0), but save logic has ability to insert zero by default if there is no value, and update logic will directly tries to increments already existing value by 1, it wont insert any value by default if its null

# Importance Of Wrapper And Primitive Types In Hibernate

If we use primitive types as variables in pojo classes and while inserting if we don’t set values for this then default primitive values will be inserted (ex for int as 0). If we use wrapper classes like Integer then null values will be inserted.

# Hibernate Lifecycle Of pojo Class Objects

Actually our POJO class object having 3 states like

* Transient state
* Persistent state
* Detached state

## Transient:

One newly created object,with out having any relation with the database, means never persistent, not associated with any Session object

## Persistent:

Having the relation with the database, associated with a unique Session object

## Detached:

previously having relation with the database [persistent ], now not associated with any Session

see the next sessions

* When ever an object of a pojo class is created then it will be in the Transient state
* When the object is in a Transient state it doesn’t represent any row of the database, i mean not associated with any Session object, if we speak more we can say no relation with the database its just an normal object
* If we modify the data of a pojo class object, when it is in transient state then it doesn’t effect on the database table
* When the object is in persistent state, then it represent one row of the database, if the object is in persistent state then it is associated with the unique Session
* if we want to move an object from persistent to detached state, we need to do either closing that session or need to clear the cache of the session
* if we want to move an object from persistent state into transient state then we need to delete that object permanently from the database

if we want to convert the object from Transient state to Persistent state we can do in 2 ways

* By saving that object like above
* By loading object from database

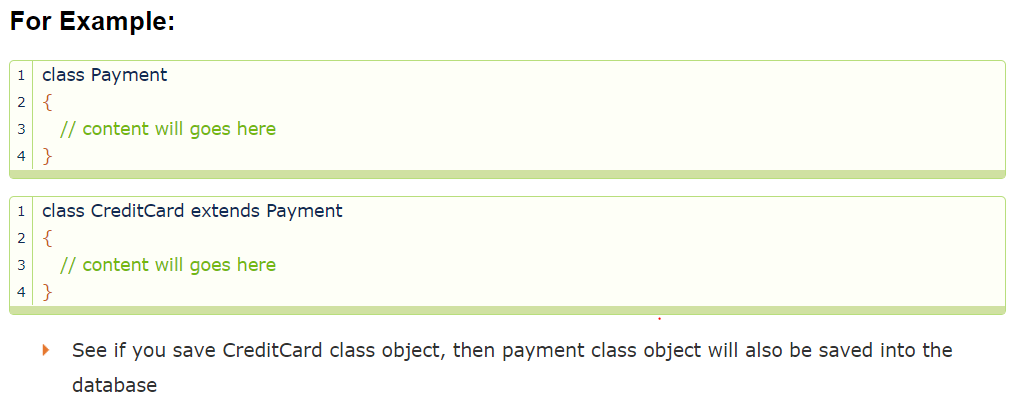
If we do any modifications all the changes will first applied to the object in session cache only (Let\_\_ we do the modifications 5 times, then 5 times we need to save the changes into the database right, which means number of round trips from our application to database will be increased, Actually if we load an object from the database, first it will saves in the cache-memory so if we do any number of changes all will be effected at cache level only and finally we can call save or update method so with the single call of save or update method the data will be saved into the database.

If we want to save an object into database then we need to call any one of the following 3 methods

* save()
* persist()
* saveOrUpdate()

# Inheritance Mapping In Hibernate – Introduction

compared to JDBC we have one main advantage in hibernate, which is hibernate inheritance.  Suppose if we have base and derived classes, now if we save derived(sub) class object, base class object will also be stored into the database.  
But the thing is we must specify in what table we need to save which object data



**Hibernate supports 3 types of Inheritance Mappings**:

* Table per class hierarchy
* Table per sub-class hierarchy
* Table per concrete class hierarchy

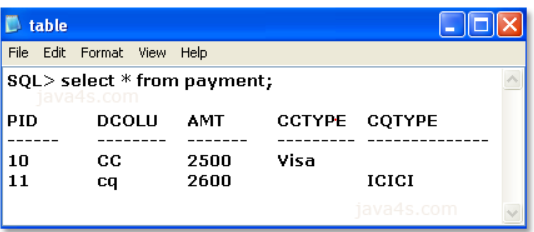
# Hibernate Inheritance: Table Per Class Hierarchy

here is the explanation and one example on hibernate table per class hierarchy, consider we have base class named Payment and 2 derived classes like CreditCard, Cheque



If we save the derived class object like CreditCard or Cheque then automatically Payment class object will also be saved into the database, and in the database all the data will be stored into a **single table** only, which is base class table for sure.

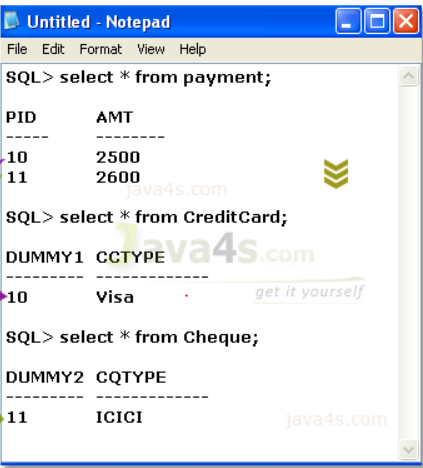
But here we must use one extra discriminator column in the database,  just to identify which derived class object we have been saved in the table along with the base class object,  if we are not using this column hibernate will throws the exception



# Hibernate Inheritance: Table Per subClass Hierarchy

x number of classes = x number of tables in the database

If we save the CreditCard class object, then first hibernate will saves the data related to super class object into the super class related table in the database and then CreditCard object data in CreditCard related table in the database, so first base class data will be saved



# Hibernate Inheritance: Table Per Concrete Class Hierarchy

x number of derived classes = x number of tables in the database

* Once we save the derived class object, then derived class data and base class data will be saved in the derived class related table in the database
* for this type we need the tables for derived classes, but not for the base class
* in the mapping file we need to use one new element <union-subclass — >under <class —>

