Hibernate HQL HCQL

18 July 2024 06:08 PM

Hibernate

List of topics:

Introduction:

- Hibernate is a java framework developed by **Gavin King**
- It is a framework which simplifies the development of java application to interact with database
- it is an open source, light weight and works based on ORM tool
- **ORM tool:**
    - **a.** Object relational mapping
    - **b.** ORM is a technique for converting the data b/w java object.
    - **c.** ORM implements responsibility of mapping the java object to relational object
    - **d.** java application -> ORM(Hibernate) -> Database
    - **e.** Some of the popular ORM tools are Hibernate,iBatis,MBatis,TopLink,etc..

Why Hibernate ?:

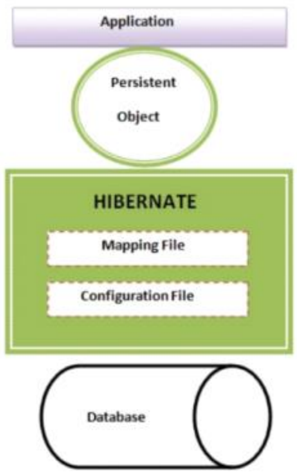
- Simplifies database interactions
- Cross database portability(Hibernate interacts with any database)

Advantages of Hibernate:

- Open source and light weight
- Fast performance
- Database independent
- Automatic database table creations
- Exception handling

Architecture:

Hibernate consists of three layer:
- Application layer (java application)
- Hibernate (mapping file , config file)
- Database (Mysql,oracle)



Hibernate arch consists of predefined objects :

- SessionFactory
- ConfigurationFactory
- TransactionFactory

POJO Class:

- POJO stands for plain old java object
- POJO is java bean
- Hibernate allows only POJO class
- POJO class consists of setter and getter

Example:

```
Class Employee  {
        private Int eid;
        private String ename;

        -> 1 Setter and 1 Getter method for eid;
        -> 1 setter and 1 getter method for ename;

        getEid() {
        }
        setEid(){
        }
        getEname() {
        }
        setEname(){
        }
}
```

Configuration File:

- The purpose of configuration file is to define the property of a database.
- Configuration file can be defined in two ways.
  - Either in XML or Annotations (XML)
- The Configuration file has to be denoted by : **hibernate.cfg.xml** (save)
- Configuration file is loaded in the Hibernate application during the runtime of an application.
- The configuration file must contain the following information:
  - Connection Properties
  - Hibernate properties
  - Mapping file resources

Note: Number of configuration files = No of databases that we are working with

Syntax:	Hibernate.cfg.xml :

```
<hibernate-configuration>
    <session-factory>

        <!Connection Properties>
            <property name="connection.driver_class">
                Load Drivers
            </property>
            <property name="connection.url">
                Connection  URL Establishment
            </property>
            <property name="connection.username">
                UserName
            </property>
            <property name="connection.password">
                Password
            </property>

        <!hibernate properties>
            <property name="show_sql">
                true/false (either can be true or false)
            </property>
            <property name="dailect">
                Database Name
            </property>
            <property name="hbm2ddl2.auto" >
                create  (creates table automatically)
            </property>

        <! Mapping files>
            <mapping resource="file(mapping)">
            <mapping resource="file(mapping)">
            <mapping resource="file(mapping)">


    </session-factory>
</hibernate-configuration>
```

Example: (**Oracle**):

Oracle:

```
<hibernate-configuration>
    <session-factory>

        <!Connection Properties>
            <property name="connection.driver_class">Oracle.jdbs.driver.OracleDriver</property>
            <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
            <property name="connection.username">system</property>
            <property name="connection.password">admin</property>

        <!hibernate properties>
            <property name="show_sql">true</property>
            <property name="dailect">oracle</property>
            <property name="hbm2ddl2.auto" >create</property>

    <! Mapping files>
        <mapping resource="employee1.hbm.xml"/>
        <mapping resource="employee2.hbm.xml"/>
        <mapping resource="employee3.hbm.xml"/>

        </session-factory>
</hibernate-configuration>
```

Example : **(MySql):**

MySql

```
<hibernate-configuration>
    <session-factory>
```

```
                <!Connection Properties>
                        <property name="connection.driver_class"> com.mysql.cf.Driver</property>
                        <property name="connection.url">jdbc:mysql://localhost:3306/klu</property>
                        <property name="connection.username">root</property>
                        <property name="connection.password">admin</property>

                <!hibernate properties>
                        <property name="show_sql">true</property>
                        <property name="dailect">mysql</property>
                        <property name="hbm2ddl2.auto" >create</property>

        <! Mapping files>
                <mapping resource="employee1.hbm.xml"/>
                <mapping resource="employee2.hbm.xml"/>
                <mapping resource="employee3.hbm.xml"/>

                </session-factory>
</hibernate-configuration>
```

Mapping File:

- It is a part of Hibernate application
- Mapping file is denoted as and can be implemented : **XML or Annotations** (XML is preferred)
- Every ORM needs a Mapping file
- It is a mechanism of placing the object properties (java object) to the specific column of the table
- This mapping file contains :
  - How a mapping can be done a POJO class to DB name and from Class properties to Column names.

POJO Class	**->**	**Table Name**
▪ Prop1	->	Column1
Prop2	->	Column2

- While creating the mapping file we can create one or multiple number of mapping files based on Application requirements

Note:

Java object	->	table

Every object will have the following properties:

a. Identity (Object name)
b. State (Object Value)
c. Behaviour (Object Method)

Syntax:

Filename.hbm.xml	<hibernate-mapping>
	    <class name=" **POJO Class Name** " table=**"Table name in DB** "/>
	        <id name=**"class-property**" column=**"column name in table**"/ >
	        <property name=**"class-propery**" column=**"column name in table**" />
	        |
	        |
	    </class>
	</hibernate-mapping>	

	Table is created by user	

Example:

Employee.hbm.xml	<hibernate-mapping>
	    <class name=" **Employee** " table=**"emp** "/>
	        <id name=**"eid**" column=**"tid**"/ >
	        <property name=**"ename**" column=**"tname**" />
	    </class>
	</hibernate-mapping>	

	Table is created by user with table name as **emp** and col name as **tid and tname**	

Note: Significance of Hibernate (Table must be created automatically)

Syntax	<hibernate-mapping>
	        <class name=" **POJO class name**" />
	                <id name=**"class-property" />**
	                <property name=**"class-property"** />
	        </class>
	</hibernate-mapping>

Example	**Employee.hbm.xml**

	<hibernate-mapping>
	        <class name=" **Employee**" />
	                <id name=**"eid"** />
	                <property name=**"ename"** />
	        </class>
	</hibernate-mapping>

Table will be created by Hibernate framework with table name as "Employee" and column name "eid and ename".

Hibernate Example : (Curd Operations):

- Every Hibernate application MUST have the following 4 files :
    - POJO Class (.java)
    - Configuration file (hibernate.cfg.xml)
    - Mapping File (filename.hbm.hml)
    - Logic file (.java file) (main method) (execute)

- Skeleton of Hibernate of Application
    - Step-1: Create a Maven Project
        - Archetypes: **maven-archetype-quickstart**
    - Step-2: pom.xml
        - Update compiler version from 1.7 to 1.8
        - Dependencies
            - Hibernate Core
            - Hibernate Entity Manager
            - Mysql
            - Oracle
    - Step-3: Update maven project
    - Step-4: Create a new folder naming as '**resources**' (config and mapping files are defined here)
        - Src->main->right click-> new-> folder -> (**resources)**
    - Step-5: Implement Hibernate Concepts
        - POJO Class (src/main/java) ->filename.java
        - Configuration file (src/main/resources) -> hibernate.cfg.xml
        - Mapping File (src/main/resources) ->filename.hbm.xml
        - Logic file (src//main/java) ->filename.java
    - Step-6: Run Logic File
        - Right click
            - Run as
                - Java application

Example-1:

Hibernate Example

(Insert)	

HQL

- HQL Stands for HIBERNATE QUERY LANGUAGE.
- HQL is database independent query language
- HQL is same a SQL, only difference is that SQL depends on the table where as HQL depends on the POJO class.
- To work with HQL, we need to use Query Interface

Query Interface :

- It is an object oriented representation of an Hibernate query
- The object of a query interface can be obtained by calling **"createQuery"** method to a session object.
- **Query q = s.createQuery("HQL");**
- The methods of query interface are :
    - executeUpdate()
    - list()
    - setFirstResult()
    - setMaxResults()
    - setParameter()

Example:

HQL Example
(to retrive all the records using ForEach)

```
package JFSDS25.JFSDS25_HQL;


import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

public class HqlRet
{

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

        SessionFactory sf = cfg.buildSessionFactory();
        Session s = sf.openSession();

        Transaction t = s.beginTransaction();

        Query<Employee> q = s.createQuery("from Employee",
Employee.class);
        List<Employee> l = q.list();

        for (Employee x : l) {
            System.out.println(x.getEname());
        }

        t.commit();
        s.close();
        sf.close();
    }
}
```

Example 2:

HQL Example
(To retrive using iterative method)

```
package JFSDS25.JFSDS25_HQL;


import java.util.Iterator;
```

```java
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

public class HQLRetIter {

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

        SessionFactory sf = cfg.buildSessionFactory();
        Session s = sf.openSession();

        Transaction t = s.beginTransaction();

        Query<Employee> q = s.createQuery("from Employee",
Employee.class);
        List<Employee> l = q.list();

        Iterator<Employee> i =l.iterator();

        while(i.hasNext()) {
           Employee e=i.next();
           System.out.println(e.getEsal());
        }

        t.commit();
        s.close();
        sf.close();
    }
}
```

Example - 3:

HQL Example
(to retrive specific range of records- **pagination**)

```java
package JFSDS25.JFSDS25_HQL;

import java.util.Iterator;
import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

public class HQLRetSpec {

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

            SessionFactory sf = cfg.buildSessionFactory();
            Session s = sf.openSession();

            Transaction t = s.beginTransaction();

            Query<Employee> q = s.createQuery("from Employee", Employee.class);
            q.setFirstResult(5);
            q.setMaxResults(15);

            t.commit();
            s.close();
            sf.close();
```

```
    }

}
```

Note : Example-1 and wx-2 are to retrive all records from table
Ex-3 is to retrive from the specific range of records (Starting record to how many num of records)

Example-4 :

To update the record: HQL Query-> update Employee set ename=:n where eid =I

q.setParameter(n,"XYZ")
q.setParameter(I,111)

```
package JFSDS25.JFSDS25_HQL;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

public class HqlUpdate {

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

        SessionFactory sf = cfg.buildSessionFactory();
        Session s = sf.openSession();

        Transaction t = s.beginTransaction();
        Query q = s.createQuery("update Employee set ename=:n
where eid=:i");
        q.setParameter("n", "saibaba");
        q.setParameter("i", 30837);
        q.executeUpdate();

        t.commit();
        s.close();
        sf.close();
    }
}
```

Example-5:

HQL Example (To delete a record)

Query--> delete from Employee where id="111"

```
package JFSDS25.JFSDS25_HQL;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

public class HQLDelete {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Configuration cfg = new Configuration();
      cfg.configure("hibernate.cfg.xml");

      SessionFactory sf = cfg.buildSessionFactory();
      Session s = sf.openSession();

      Transaction t = s.beginTransaction();
      Query q = s.createQuery("delete from Employee where eid =
30837");

        q.executeUpdate();

        t.commit();
        s.close();
        sf.close();


    }

}
```

Example-6:

HQL Example to insert a record

Query -> Insert into Employee(eid,ename,esal) values (111,"querty",85254);

```

package JFSDS25.JFSDS25_HQL;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

public class HQLInsert {

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

        SessionFactory sf = cfg.buildSessionFactory();
        Session s = sf.openSession();

        Transaction t = s.beginTransaction();
        Query q = s.createSQLQuery("insert into
Employee(eid,ename,esal) values (111,'querty',85254)");

        q.executeUpdate();

        t.commit();
        s.close();
        sf.close();
           // TODO Auto-generated method stub

     }

}
```

Example - 7:

HQL example to retrive all records with partial number of columns

Eid,ename,esal ->Employee
Eid,Ename - > HQL

```

package JFSDS25.JFSDS25_HQL;
import java.util.Iterator;
import java.util.List;

import org.hibernate.~~Query~~;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;


public class HQLsplRet {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
				Configuration cfg  = new 
Configuration();
				cfg.configure("hibernate.cfg.xml");
				
				SessionFactory sf= 
cfg.buildSessionFactory();
				Session s=sf.openSession();
				
				Transaction t= s.beginTransaction();
				
				~~Query~~<Object[]> q = 
s.createQuery("select eid,ename from Employee");
				
				List<Object[]>l=q.~~list~~();
				Iterator<Object[]> i= l.iterator();
				while(i.hasNext()) {
					Object ob[]=i.next();
					System.out.println(ob[0]+" 
"+ob[1]);
				}
				
		// TODO Auto-generated method stub

	}

}
```

Example 8:

(to retrive all in one column)

```
package JFSDS25.JFSDS25_HQL;

import java.util.Iterator;
import java.util.List;

import org.hibernate.~~Query~~;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class HQLRetAllclm {

	public static void main(String[] args) {
	Configuration cfg  = new Configuration();
	cfg.configure("hibernate.cfg.xml");
	
	SessionFactory sf= cfg.buildSessionFactory();
	Session s=sf.openSession();
	
	Transaction t= s.beginTransaction();
	
	~~Query~~<Object[]> q = s.createQuery("select eid from Employee");
	
	List l=q.~~list~~();
	Iterator i= l.iterator();
	while(i.hasNext()) {
		Object ob=(Object)i.next();
		System.out.println(ob);
	}
	
		// TODO Auto-generated method stub

	}
```

	}

HCQL:

- Hibernate Criteria Query Language
- Using HCQL, we can able to retrive records with all number of columns every time
- To implement HCQL in hibernate application we need to use Criteria interface

Criteria Interface:

- Criteria interface object can be obtained by calling createCriteria() method to a session object
- **Criteria cr = s.createCriteria();**
- Methods of criteria interface are:
    - Add()
    - addOrder()
    - setFirstResults()
    - setMaxResults()
    - list()

Restriction Class:

- In order to have restriction/condition/criteria on HQL query
- The Methods of restriction class are
    - lt()
    - le()
    - gt()
    - ge()
    - eq()
    - neq()

Order class:

- Consists of many methods, in order to display or retrive the data from the database table. Either in ascending or descinding order.
- The methods are:
    - asc()
    - dsc()

Example-1	

HCQL EXAMPLE
(to retrive records using foreach loop)
	package JFSDS25.JFSDS25_HQL;
	
	import java.util.List;
	
	import org.hibernate.Criteria;
	import org.hibernate.Session;
	import org.hibernate.SessionFactory;
	import org.hibernate.Transaction;
	import org.hibernate.cfg.Configuration;
	
	public class HCQLRet {
	
	    public static void main(String[] args)
	    {
	        Configuration cfg = new Configuration();
	   cfg.configure("hibernate.cfg.xml");
	
	   SessionFactory sf = cfg.buildSessionFactory();
	   Session s = sf.openSession();
	
	   Transaction t = s.beginTransaction();
	
	   Criteria cr= s.createCriteria(Employee.class);
	
	   List<Employee> l = cr.list();

```
        for (Employee x : l) {
           System.out.println(x.getEname());
        }

        t.commit();
        s.close();
        sf.close();

          }

}
```

Example - 2: Iterator

```
package JFSDS25.JFSDS25_HQL;

import java.util.Iterator;
import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

public class HCQLRetItr {

      public static void main(String[] args)
      {
            // TODO Auto-generated method stub
             Configuration cfg = new Configuration();
              cfg.configure("hibernate.cfg.xml");

              SessionFactory sf = cfg.buildSessionFactory();
              Session s = sf.openSession();

              Transaction t = s.beginTransaction();

              Criteria cr = s.~~createCriteria~~(Employee.class);
              List<Employee> l = cr.list();

              Iterator<Employee> i =l.iterator();

              while(i.hasNext()) {
                 Employee e=i.next();
                 System.out.println(e.getEname());
              }

              t.commit();
              s.close();
              sf.close();

      }

}
```

Example-3 : Padination

```
package JFSDS25.JFSDS25_HQL;

import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
```

```java
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

public class HCQLRetSpec {

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

		 SessionFactory sf = cfg.buildSessionFactory();
		 Session s = sf.openSession();

		 Transaction t = s.beginTransaction();

		 Criteria cr = s.~~createCriteria~~(Employee.class);
		 cr.setFirstResult(5);
		 cr.setMaxResults(15);
		 List<Employee> l = cr.list();

		 for (Employee x : l) {
		     System.out.println(x.getEname());
		 }

		 t.commit();
		 s.close();
		 sf.close();

		 t.commit();
		 s.close();
		 sf.close();

	}

}
```

Example-4

Restriction class

```java
package JFSDS25.JFSDS25_HQL;

import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.criterion.Restrictions;

public class HCQLRestriction {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Configuration cfg = new Configuration();
	     cfg.configure("hibernate.cfg.xml");

	     SessionFactory sf = cfg.buildSessionFactory();
	     Session s = sf.openSession();

	     Transaction t = s.beginTransaction();

	     Criteria cr = s.~~createCriteria~~(Employee.class);
	    cr.add((Restrictions.gt("esal", 100.0)));
	    List<Employee> l = cr.list();
```

```
        for (Employee x : l) {
            System.out.println(x.getEsal());
        }

        t.commit();
        s.close();
        sf.close();


    }

}
```

Example-5:

Order class

```
package JFSDS25.JFSDS25_HQL;

import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Restrictions;

public class HCQLOrder {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Configuration cfg = new Configuration();
      cfg.configure("hibernate.cfg.xml");

      SessionFactory sf = cfg.buildSessionFactory();
      Session s = sf.openSession();

      Transaction t = s.beginTransaction();

      Criteria cr = s.~~createCriteria~~(Employee.class);
     cr.add((Restrictions.gt("esal", 100.0)));
     cr.addOrder(Order.asc("esal"));
     List<Employee> l = cr.list();

     for (Employee x : l) {
         System.out.println(x.getEsal());
     }

     t.commit();
     s.close();
     sf.close();


    }

}
```

Inheritance Mapping :

- **Inheritance:**
    - Getting the properties from the Base class to the Derived class refers to the inheritance.
    - Inheritances can be of 5 types (OOP)
        - Single Inheritance (1 base class[A], 1 Derived class [B]   A-->B)
        - Multiple Inheritance (Multiple Base classes A,B,C , 1 Derived Class D :   Classes A,B,C ---> D)
        - Multilevel Inheritance (1 Base class A, another Base class B, and soo on..... : A-->B-->C c aquires prop of b, b

aquries prop of a.)

- Heirarical inheritance (1 Base class A, Multiple Derived classes B,C,D.... : A-----> B,C,D....)
- Hybrid Inheritance (Combination of two inheritances **Multiple & Hirarichal inheritance**)
  - A---------> B,C,D ---------> E
    - Java does not supports the Multiple Inheritance

- **<u>Mapping :</u>**
  - Refers to the relationship between the different tables in the database. Employee, Product.....
  - Different types of Mappings/Relations are:
    - IS - A (Inheritance)
    - HAS - A (Association)

- Inheritance Mapping can be Implemented in Three ways in Hibernate application:
  - Table Per Class
  - Table Per Sub-Class
  - Table Per Concrete-Class
-

Scenario-1:	**Payment** **|** ----------------------------- **|** **|** **Card** **Cheque**
Scenario-2	**Person** **|** ------------------------- **|** **|** **Student** **Employee**

1. <u>Table Per class</u>
   i. Payment ii. Person
2. <u>Table Per Sub Class</u>
   i. Card , Cheque ii. Student, Employee
3. <u>Table Per Concrete Class</u>
   i. Payment , Card, Cheque ii. Person , Student, Employee

Table Per Class:	Create a POJO Class src/main/java (**PAYMENT)**

	Class Payment{
	    int pid;
	    double pamount;
	--> generate setters and getters
	
	}

	Class Card extends Payment{
	        String cardType;
	        --> Generate getters and setters
	}

	Class Cheque exteends Payment{
	        Strin chequeType;
	        --> generate setters and getters
	}

Configuration file:	Hiberanate properties, Connection prperties and Mapping properties

Mapping
File(payment.hbm.xml)
	<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
	<hibernate-mapping>
	  <class name="JFSDS25_IMTC.JFSDS25_IMTC.Payment" >
	    <id name="pid" />
	    <property name="pamount" />

	```
	  </class>
	  <subclass name="JFSDS25_IMTC.JFSDS25_IMTC.Card" discriminator-value="c">
	    <property name="cardType"/>
	  </subclass>
	  <subclass name="JFSDS25_IMTC.JFSDS25_IMTC.Cheque" discriminator-value="cq">
	    <property name="chequeType"/>
	  </subclass>
	</hibernate-mapping>
	```

Logic File(TablePerClass)	
	```
	Class TablePerClass
	        {
	        Configuration cf = new Configuration();
	        SessionFactory sf = cf.createSessionFactory();
	        Session s = sf.openSession();
	        Transaction t=s.beginTransaction();
	        Card c=new Card();
	        Cheque cq= new Cheque();
	        c.setPid("101");
	        c.setPamount(145000);
	        c.setCardType("Credit Card");

	        Cq.setPid(201);
	        Cq.setPamount(145820);
	        Cq.setChequeType("RTGS");

	        s.save(c);
	        s.save(cq);
	        t.commit();
	        s.close();
	        Sc.close();

	        }
	```

	DB: Payment: pid , pamount , cardType, chequeType

Example- 2	POJO Class is same
TablePerSubClass	Configuration file is same as above

Mapping File:	
	```
	<hibernate-mapping>
	        <class name="com.klu.JFSDS25_IMTC.Payment">
	                <id name="pid" />
	                <property name="pamount" />
	        </class>
	        <joinedclass name="com.klu.JFSD_IMTC" discriminator="c" >

	                <property name="cardType"/>
	        </joinedclass>
	        <joinedClass name="com.klu.JFSD25_IMTC.Cheque" discriminator="cq">
	                <property name=chequeType""/>
	        </joinedclass>

	</hibernate-mapping>
	```

Logic File:	Same as previous
	DB-> Table Per Subclass Card Cheque pid pamount cardType pid pamount cardType

TablePerConcreteClass	POJO Class is same

	Configuration file is same as above

Mapping File:	`<hibernate-mapping>`
	`    <class name="com.klu.JFSDS25_IMTC.Payment">`
	`        <id name="pid" />`
	`        <property name="pamount" />`
	`    </class>`
	`    <unionclassname="com.klu.JFSD_IMTC" discriminator="c" >`
	
	`        <property name="cardType"/>`
	`    </unionclass>`
	`    <unionclass name="com.klu.JFSD25_IMTC.Cheque" discriminator="cq">`
	`        <property name=chequeType""/>`
	`    </unionclass>`
	`</hibernate-mapping>`
	
	`DB-> Payment                    Card              Cheque`
	`     ------------           ----------------  --------------------`
	`     Pid, pamount                 cardType      chequeType`