

Unit 3. ACCESS USING OBJECT- RELATIONAL MAPPING (ORM)

Part 3. Hibernate using HQL

Acceso a Datos (ADA) (a distancia en inglés)

CFGs Desarrollo de Aplicaciones Multiplataforma (DAM)

Abelardo Martínez

Year 2024-2025

Credits



- Notes made by Abelardo Martínez.
- Based and modified from Sergio Badal (www.sergiobadal.com).
- The images and icons used are protected by the [LGPL](#) licence and have been obtained from:
 - https://commons.wikimedia.org/wiki/Crystal_Clear
 - <https://www.openclipart.org>

Contents

1. HIBERNATE QUERIES
2. USING NATIVE QUERIES
3. HQL QUERIES
 1. Using HQL queries
 2. Upgrading example with HQL
4. HQL CRITERIA
 1. Using HQL criteria
 2. Upgrading example with HQL criteria
5. ACTIVITIES FOR NEXT WEEK
6. BIBLIOGRAPHY



1. HIBERNATE QUERIES

Hibernate queries



Hibernate provides several options to deal with our database:

- 1) **Native SQL queries (low level of abstraction)**
 - Use MAINLY native SQL language to set complex queries
- 2) **HQL Queries (medium level of abstraction)**
 - MIX specific Hibernate query language and methods
- 3) **HQL Queries using criteria (high level of abstraction)**
 - Use MAINLY specific methods (**NO QUERY AT ALL!**)

2. USING NATIVE QUERIES

Hibernate native queries

- Hibernate provides an option to execute native SQL queries through the use of SQLQuery object.
- Hibernate SQL Query is **very handy** when we have to execute database vendor **specific queries** that are not supported by Hibernate API.
- For **normal scenarios**, **Hibernate SQL query is not the recommended approach** because we lose benefits related to hibernate association and hibernate first level **cache**.
- The syntax is as follows:

```
query = session.createSQLQuery("select ... from ... where ... order by ...");  
rows = query.list();  
for(Object[] row : rows){ ... }
```

Detailed example: <https://www.journaldev.com/3422/hibernate-native-sql-query-example>

Example 1

For instance, you can join two tables by using direct SQL queries:

```
query = session.createSQLQuery("select e.emp_id, emp_name, emp_salary,address_line1, city,  
    zipcode from Employee e, Address a where a.emp_id=e.emp_id");  
rows = query.list();  
for(Object[] row : rows){  
    Employee emp = new Employee();  
    emp.setId(Long.parseLong(row[0].toString()));  
    emp.setName(row[1].toString());  
    emp.setSalary(Double.parseDouble(row[2].toString()));  
    Address address = new Address();  
    address.setAddressLine1(row[3].toString());  
    address.setCity(row[4].toString());  
    address.setZipcode(row[5].toString());  
    emp.setAddress(address);  
    System.out.println(emp);  
}
```


Example 2

Additionally, you can add parameters:

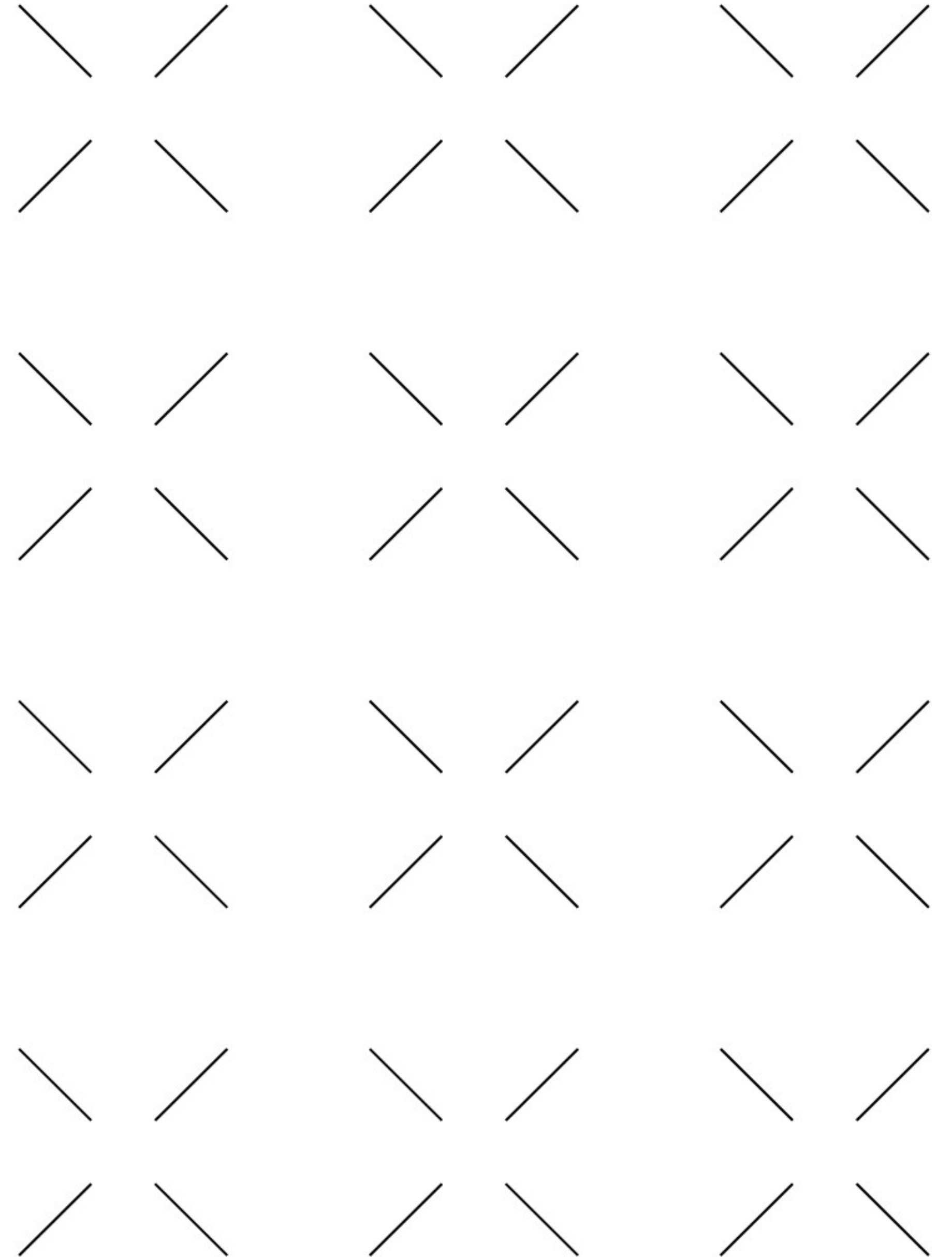
```
query = session
    .createQuery("select emp_id, emp_name, emp_salary from Employee where emp_id = ?");
List<Object[]> empData = query.setLong(0, 1L).list();
for (Object[] row : empData) {
    Employee emp = new Employee();
    emp.setId(Long.parseLong(row[0].toString()));
    emp.setName(row[1].toString());
    emp.setSalary(Double.parseDouble(row[2].toString()));
    System.out.println(emp);
}
```

To sum up

- That was a brief introduction of Hibernate Native SQL Query, you should avoid using it unless you want to execute any database specific queries.
- For a deep view on those queries you can have a look here:
<https://docs.jboss.org/hibernate/core/3.3/reference/en/html/querysql.html>

3. HQL QUERIES

3.1 Using HQL queries



Hibernate Query Language (HQL)

- Hibernate Query Language (HQL) is an **object-oriented query language, similar to SQL**, but instead of operating on tables and columns, HQL works with persistent objects and their properties.
- **HQL queries are translated by Hibernate into conventional SQL queries**, which in turns perform action on database.
- Although you can use native SQL, **it is recommended to use HQL whenever possible** to avoid database portability hassles, and to take advantage of Hibernate's caching strategies.
- The syntax is as follows:

```
List items = session.createQuery("from ... where ... order by ...").list();  
for (Iterator iterator = items.iterator(); iterator.hasNext();) { ... }
```



Be aware of the keywords like SELECT, FROM, and WHERE, etc., are not case sensitive, but properties like table and column names are case sensitive in HQL.

SELECT / FROM

FROM Clause

You will use **FROM** clause if you want to load a complete persistent objects into memory. Following is the simple syntax of using FROM clause –

```
String hql = "FROM Employee";  
Query query = session.createQuery(hql);  
List results = query.list();
```



Used so far in our
code (last week's)

SELECT Clause

The **SELECT** clause provides more control over the result set then the from clause. If you want to obtain few properties of objects instead of the complete object, use the SELECT clause. Following is the simple syntax of using SELECT clause to get just first_name field of the Employee object –

```
String hql = "SELECT E.firstName FROM Employee E";  
Query query = session.createQuery(hql);  
List results = query.list();
```



New

It is notable here that **Employee.firstName** is a property of Employee object rather than a field of the EMPLOYEE table.

WHERE / ORDER

WHERE Clause

If you want to narrow the specific objects that are returned from storage, you use the WHERE clause. Following is the simple syntax of using WHERE clause –

```
String hql = "FROM Employee E WHERE E.id = 10";  
Query query = session.createQuery(hql);  
List results = query.list();
```



New

ORDER BY Clause

To sort your HQL query's results, you will need to use the **ORDER BY** clause. You can order the results by any property on the objects in the result set either ascending (ASC) or descending (DESC). Following is the simple syntax of using ORDER BY clause –

```
String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";  
Query query = session.createQuery(hql);  
List results = query.list();
```



New

If you wanted to sort by more than one property, you would just add the additional properties to the end of the order by clause, separated by commas as follows –

```
String hql = "FROM Employee E WHERE E.id > 10 " +  
            "ORDER BY E.firstName DESC, E.salary DESC ";  
Query query = session.createQuery(hql);  
List results = query.list();
```

GROUP BY

GROUP BY Clause

This clause lets Hibernate pull information from the database and group it based on a value of an attribute and, typically, use the result to include an aggregate value. Following is the simple syntax of using GROUP BY clause –

```
String hql = "SELECT SUM(E.salary), E.firstName FROM Employee E " +  
            "GROUP BY E.firstName";  
Query query = session.createQuery(hql);  
List results = query.list();
```

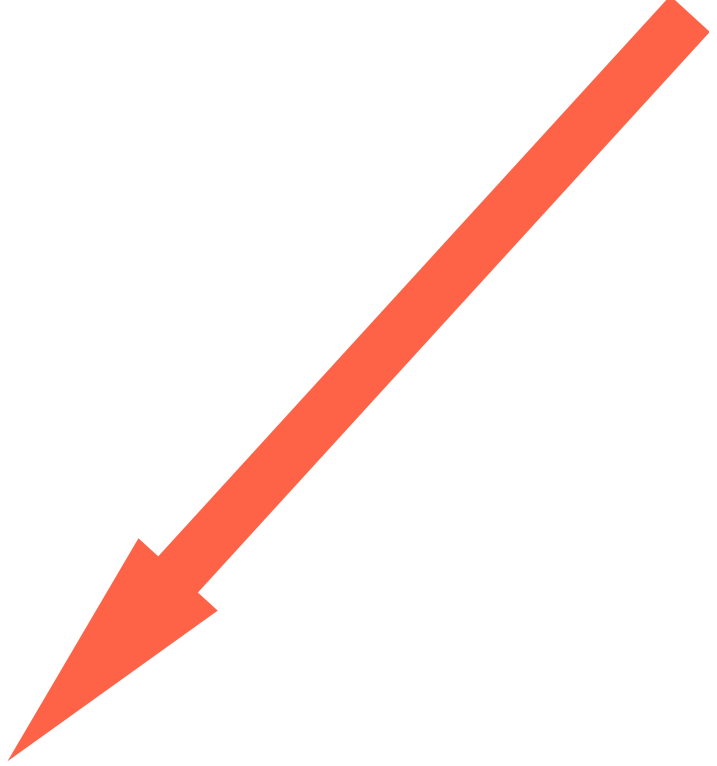
 New

- You can also use complex DML sentences (INSERT, UPDATE, DELETE) as you can check here:
https://www.tutorialspoint.com/hibernate/hibernate_query_language.htm

AGGREGATE FUNCTIONS

- You may call avg(), min(), max() etc. aggregate functions by HQL.
- Let's see some common examples:

Be aware of the way we should get these data



For further information: <https://www.javatpoint.com/hql>

Example to get total salary of all the employees

```
Query q=session.createQuery("select sum(salary) from Emp");  
List<Integer> list=q.list();  
System.out.println(list.get(0));
```

Example to get maximum salary of employee

```
Query q=session.createQuery("select max(salary) from Emp");
```

Example to get minimum salary of employee

```
Query q=session.createQuery("select min(salary) from Emp");
```

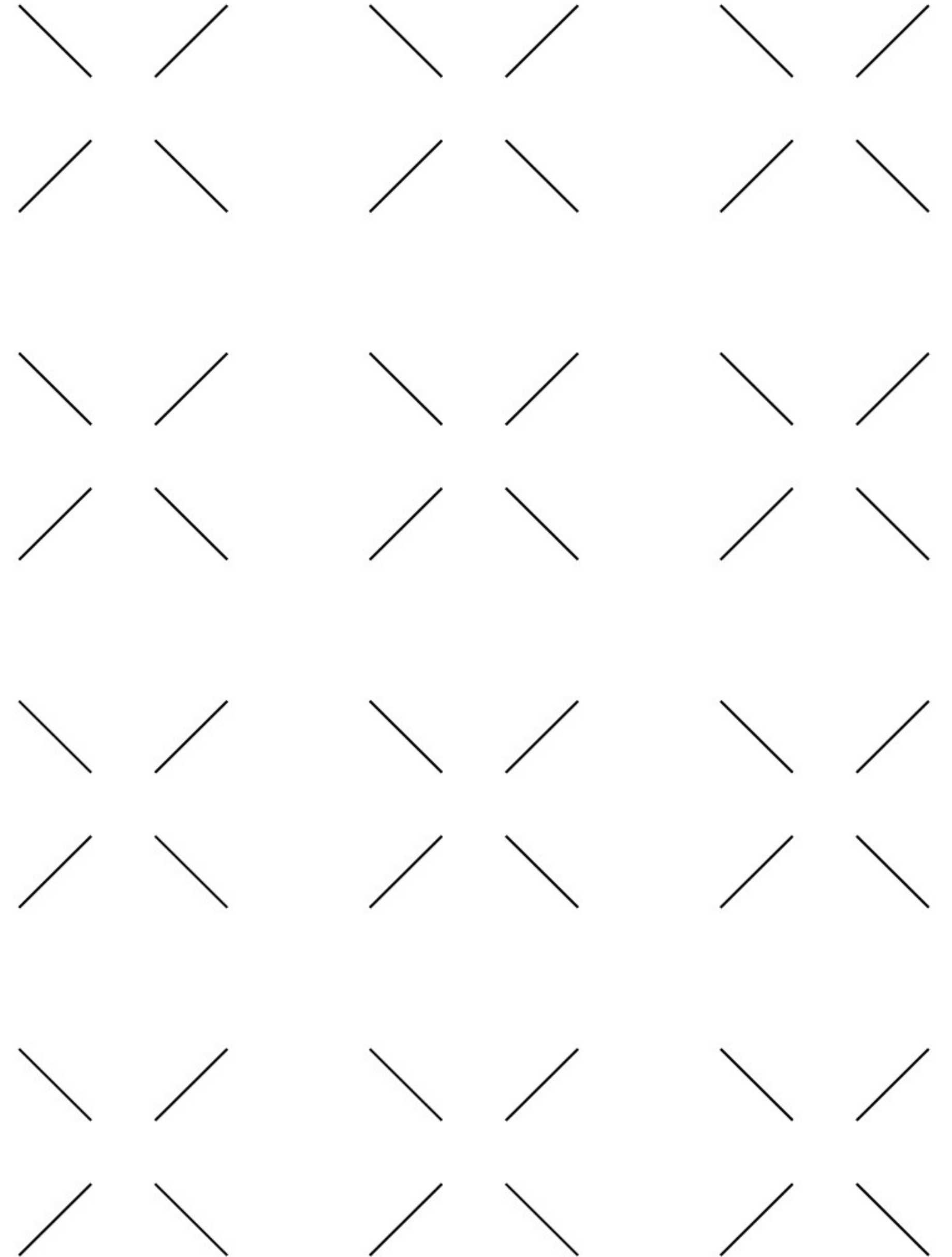
Example to count total number of employee ID

```
Query q=session.createQuery("select count(id) from Emp");
```

Example to get average salary of each employees

```
Query q=session.createQuery("select avg(salary) from Emp");
```

3.2 Upgrading example with HQL



Upgrading to HQL queries. Main programme

```
public class TestHibernateMySQL {

    // MAIN PROGRAMME
    public static void main(String[] stArgs) {

        //Create new objects DAO for CRUD operations
        EmployeeDAO objEmployeeDAO = new EmployeeDAO();
        CertificateDAO objCertificateDAO = new CertificateDAO();

        //TRUNCATE TABLES. Delete all records from the tables
        objEmployeeDAO.deleteAllItems();
        objCertificateDAO.deleteAllItems();

        /* Add records in the database */
        Certificate objCert1 = objCertificateDAO.addCertificate("MBA");
        Certificate objCert2 = objCertificateDAO.addCertificate("PMP");

        //Set of certificates
        HashSet<Certificate> hsetCertificates = new HashSet<Certificate>();
        hsetCertificates.add(objCert1);
        hsetCertificates.add(objCert2);

        /* Add records in the database */
        Employee objEmp1 = objEmployeeDAO.addEmployee("Alfred", "Vincent", 15000, hsetCertificates);
        Employee objEmp2 = objEmployeeDAO.addEmployee("John", "Gordon", 8000, hsetCertificates);
        Employee objEmp3 = objEmployeeDAO.addEmployee("Edgard", "Codd", 20000, hsetCertificates);
        Employee objEmp4 = objEmployeeDAO.addEmployee("Joseph", "Smith", 5000, hsetCertificates);
        Employee objEmp5 = objEmployeeDAO.addEmployee("Mary", "Jones", 15000, hsetCertificates);

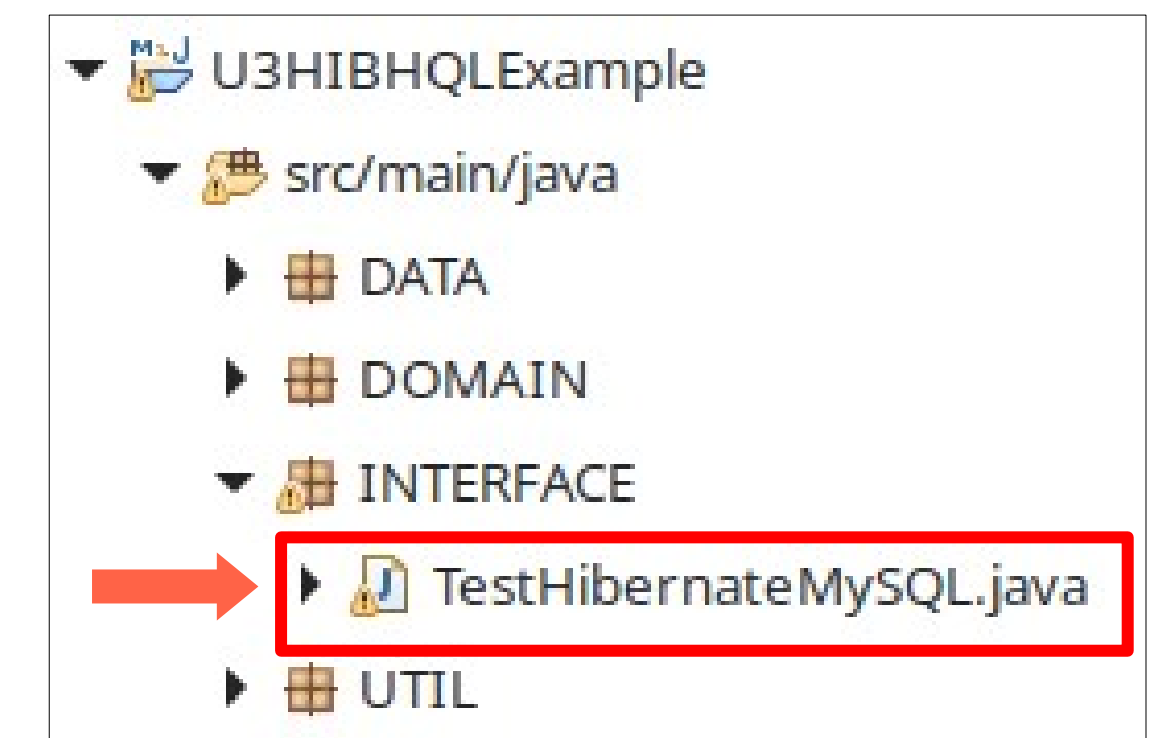
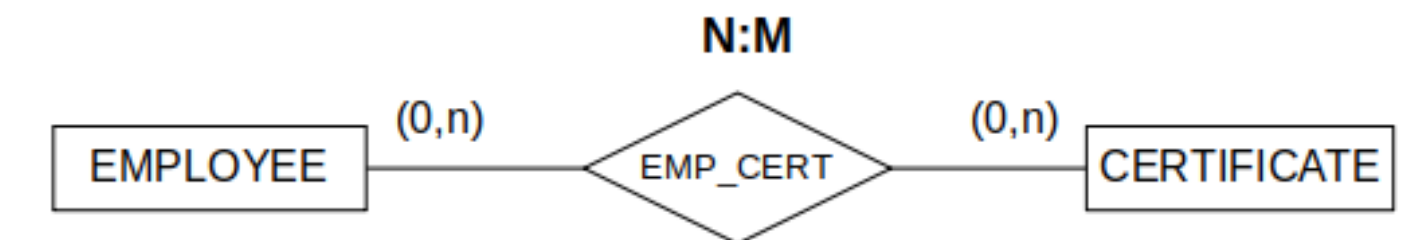
        /* List down all the employees */
        objEmployeeDAO.listEmployees();
        objEmployeeDAO.listRichEmployees();

        //Close global hibernate session factory
        HibernateUtil.shutdownSessionFactory();
    }
}
```

← Last week's

← This week's

We'll create a new method to list only the employees earning > 10000, order by salary desc




Upgrading to HQL queries. Method

It's as easy as adding the condition using HQL, almost identical (in this case) to standard SQL

```
/* Method to READ all the employees earning > 10000 */
public void listRichEmployees() {

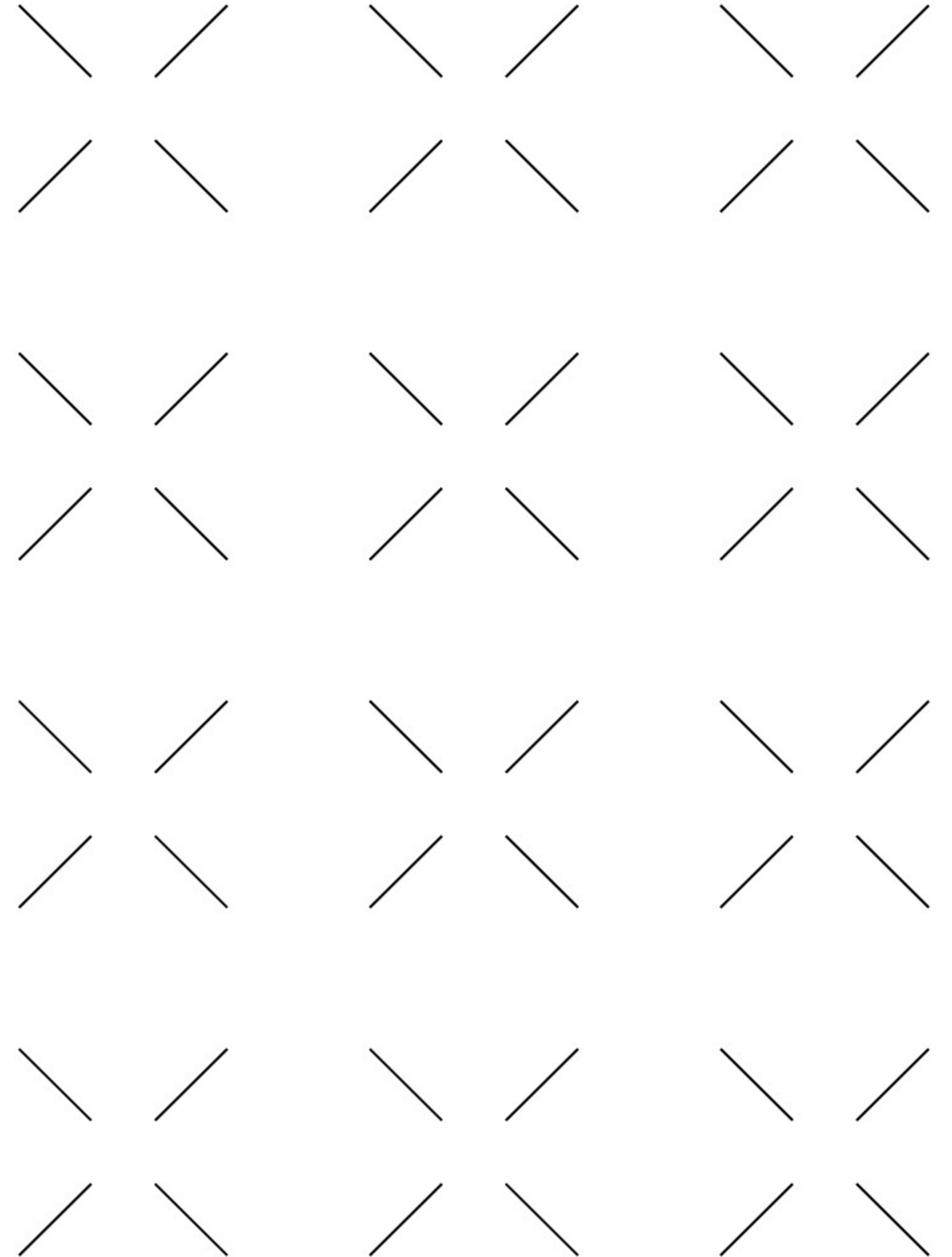
    Session hibSession = HibernateUtil.SFACTORY.openSession(); //open hibernate session factory
    Transaction txDB = null; //database transaction

    try {
        txDB = hibSession.beginTransaction(); //starts transaction
        List<Employee> listEmployees = hibSession.createQuery("FROM Employee WHERE dSalary > 10000 ORDER BY dSalary DESC", Employee.class).list();
        if (listEmployees.isEmpty())
            System.out.println("***** No items found");
        else {
            System.out.println("\n***** Start listing ...\n");
            for (Iterator<Employee> itEmployee = listEmployees.iterator(); itEmployee.hasNext();) {
                Employee objEmployee = (Employee) itEmployee.next();
                System.out.print("First Name: " + objEmployee.getstFirstName() + " | ");
                System.out.print("Last Name: " + objEmployee.getstLastName() + " | ");
                System.out.println("Salary: " + objEmployee.getdSalary());
                Set<Certificate> setCertificates = objEmployee.getrelCertificates();
                for (Iterator<Certificate> itCertificate = setCertificates.iterator(); itCertificate.hasNext();) {
                    Certificate objCertificate = (Certificate) itCertificate.next();
                    System.out.println("Certificate: " + objCertificate.getstCertName());
                }
            }
        }
        txDB.commit(); //ends transaction
    } catch (HibernateException hibe) {
        if (txDB != null)
            txDB.rollback(); //something went wrong, so rollback
        hibe.printStackTrace();
    } finally {
        hibSession.close(); //close hibernate session
    }
}
```



4. HQL CRITERIA

4.1 Using HQL criteria



HQL Criteria

- The Hibernate Criteria Query Language (HCQL) is used to fetch the records based on the specific criteria.
- The Criteria interface provides many methods to specify criteria.

The object of Hibernate Criteria `createCriteria()` is deprecated since Hibernate 5 (sep 2015), you should use **JPA CriteriaQuery** instead.

<https://stackoverflow.com/questions/40720799/deprecated-createcriteria-method-in-hibernate-5>



- Most of the resources you find surfing the Net will be about the deprecated version. You can use this ones to use the new approach:

<https://www.baeldung.com/hibernate-criteria-queries>

Hibernate-specific extensions to the Criteria API

- JPA specification defines the string-based JPQL query language.
- Hibernate extends it (JPA) to support things like database-specific functions, window functions, and set-based operations.
- Since version 6, Hibernate has done the same for JPA's Criteria API.
- The syntax is as follows:

```
CriteriaBuilder cb = session.getCriteriaBuilder();
```

```
CriteriaQuery<Item> cr = cb.createQuery(Item.class);
```

```
cr.select(root).where(cb.gt(root.get("field"), 10000));  “field” must have same type as the other value  
cr.orderBy(cb.desc(root.get("field")));
```

```
Query<Item> query = session.createQuery(cr);
```

```
List<Item> items = query.getResultList();
```

```
for (Iterator iterator = items.iterator(); iterator.hasNext();) { ... }
```

For further information: <https://thorben-janssen.com/hibernate-specific-extensions-to-the-criteria-api/>

Upgrading to HQL Criteria. Method

```
public void listEmployeesHQLC() {  
  
    Session hibSession = HibernateUtil.SFACTORY.openSession(); //open hibernate session factory  
    Transaction txDB = null; //database transaction  
    System.out.println("\n***** Listing employees using HQL criteria...\n");  
  
    try {  
        txDB = hibSession.beginTransaction(); //starts transaction  
        // 1. Create a CriteriaBuilder instance by calling the Session.getCriteriaBuilder() method.  
        CriteriaBuilder crbCritBuilder = hibSession.getCriteriaBuilder();  
        // 2. Create a query object by creating an instance of the CriteriaQuery interface.  
        CriteriaQuery<Employee> crqHQL = crbCritBuilder.createQuery(Employee.class);  
        // 3. Set the query Root by calling the from() method on the CriteriaQuery  
        // object to define a range variable in FROM clause.  
        Root<Employee> rootEmployee = crqHQL.from(Employee.class);  
        // 4. Specify what the type of the query result will be by calling the select()  
        // method of the CriteriaQuery object.  
        crqHQL.select(rootEmployee);  
        // 5. Prepare the query for execution by creating a org.hibernate.query.Query  
        // instance by calling the Session.createQuery() method, specifying the type of the query result.  
        Query<Employee> qryHQL = hibSession.createQuery(crqHQL);  
        // 6. Execute the query by calling the getResultList() or getSingleResult()  
        // method on the org.hibernate.query.Query object.  
        List<Employee> lstEmployee = qryHQL.getResultList();  
        if (lstEmployee.isEmpty())  
            System.out.println("***** No items found");  
        else {  
            for (Iterator<Employee> itEmployee = lstEmployee.iterator(); itEmployee.hasNext();) {  
                Employee objEmployee = (Employee) itEmployee.next();  
                System.out.print("First Name: " + objEmployee.getstFirstName() + " | ");  
                System.out.print("Last Name: " + objEmployee.getstLastName() + " | ");  
                System.out.println("Salary: " + objEmployee.getdSalary());  
                Set<Certificate> relCertificates = objEmployee.getrelCertificates();  
                for (Iterator<Certificate> itCertificate = relCertificates.iterator(); itCertificate.hasNext();) {  
                    Certificate objCertificate = (Certificate) itCertificate.next();  
                    System.out.println("Certificate: " + objCertificate.getstCertName());  
                }  
            }  
        }  
        txDB.commit(); //ends transaction  
    } catch (HibernateException hibe) {  
        if (txDB != null)  
            txDB.rollback(); //something went wrong, so rollback  
        hibe.printStackTrace();  
    } finally {  
        hibSession.close(); //close hibernate session  
    }  
}
```

These are the standard instructions to get all elements from a class.



Comparison operators

And these, the refinements we can apply to get specific data.

In order to get items having a price of more than 1000:

```
cr.select(root).where(cb.gt(root.get("itemPrice"), 1000));
```

Next, getting items having *itemPrice* less than 1000:

```
cr.select(root).where(cb.lt(root.get("itemPrice"), 1000));
```

Items having *itemName* contain *Chair*:

```
cr.select(root).where(cb.like(root.get("itemName"), "%chair%"));
```

Records having *itemPrice* between 100 and 200:

```
cr.select(root).where(cb.between(root.get("itemPrice"), 100, 200));
```

Items having *itemName* in *Skate Board*, *Paint* and *Glue*:

```
cr.select(root).where(root.get("itemName").in("Skate Board", "Paint", "Glue"));
```

Property NULL/NOT NULL

And these, the refinements we can apply to get specific data.

To check if the given property is null:

```
cr.select(root).where(cb.isNull(root.get("itemDescription")));
```

To check if the given property is not null:

```
cr.select(root).where(cb.isNotNull(root.get("itemDescription")));
```

We can also use the methods *isEmpty()* and *isNotEmpty()* to test if a *List* within a class is empty or not.

Logical operators and chain expressions

We can also use the methods *isEmpty()* and *isNotEmpty()* to test if a *List* within a class is empty or not.

Additionally, we can combine two or more of the above comparisons. **The Criteria API allows us to easily chain expressions:**

```
Predicate[] predicates = new Predicate[2];
predicates[0] = cb.isNull(root.get("itemDescription"));
predicates[1] = cb.like(root.get("itemName"), "chair%");
cr.select(root).where(predicates);
```

And these, the refinements we can apply to get specific data.

To add two expressions with logical operations:

```
Predicate greaterThanPrice = cb.gt(root.get("itemPrice"), 1000);
Predicate chairItems = cb.like(root.get("itemName"), "Chair%");
```

Items with the above-defined conditions joined with *Logical OR*.

```
cr.select(root).where(cb.or(greaterThanPrice, chairItems));
```

To get items matching with the above-defined conditions joined with *Logical AND*.

```
cr.select(root).where(cb.and(greaterThanPrice, chairItems));
```

Aggregate functions

And these, the refinements we can apply to get specific data.

Now let's see the different aggregate functions.

Get row count:

```
CriteriaQuery<Long> cr = cb.createQuery(Long.class);
Root<Item> root = cr.from(Item.class);
cr.select(cb.count(root));
Query<Long> query = session.createQuery(cr);
List<Long> itemProjected = query.getResultList();
```

The following is an example of aggregate functions — *Aggregate* function for *Average*:

```
CriteriaQuery<Double> cr = cb.createQuery(Double.class);
Root<Item> root = cr.from(Item.class);
cr.select(cb.avg(root.get("itemPrice")));
Query<Double> query = session.createQuery(cr);
List avgItemPriceList = query.getResultList();
```

Other useful aggregate methods are *sum()*, *max()*, *min()*, *count()*, etc.

ORDER BY

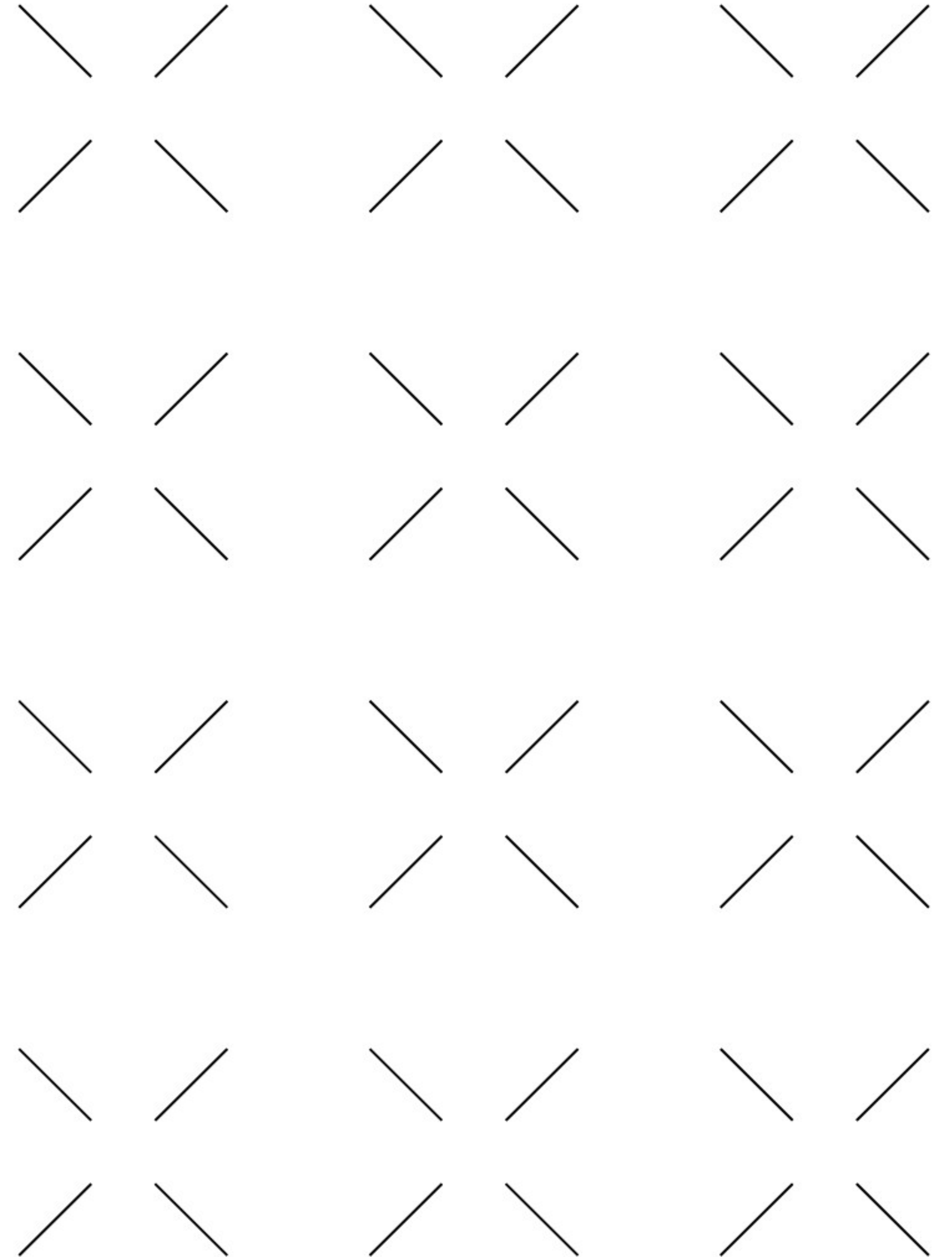
And these, the refinements we can apply to get specific data.

In the following example, we order the list in ascending order of the name and then in descending order of the price:

```
cr.orderBy(  
    cb.asc(root.get("itemName")),  
    cb.desc(root.get("itemPrice")));
```

You can also use complex DML sentences (INSERT, UPDATE, DELETE) and use predicates as you can check here: <https://www.baeldung.com/hibernate-criteria-queries>

4.2 Upgrading example with HQL criteria



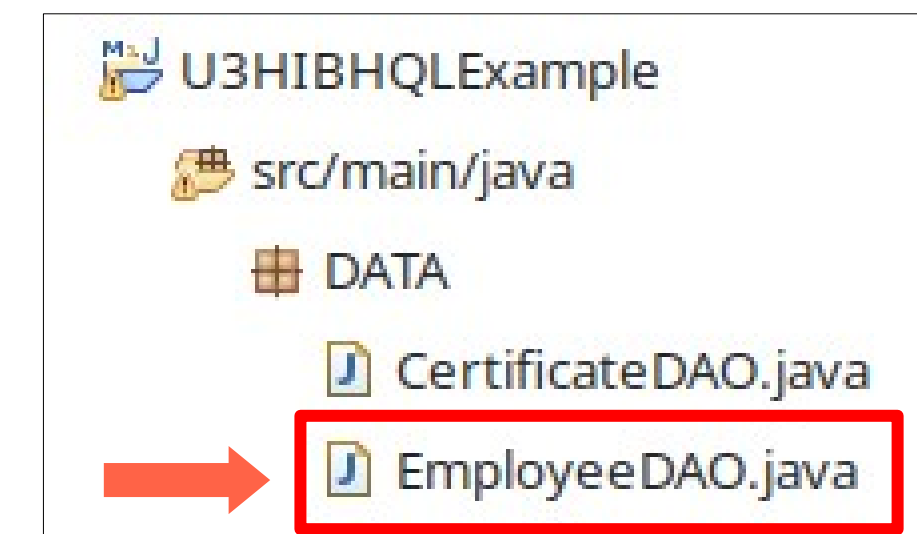
Upgrading to HQL Criteria. Method

```
public void listRichEmployeesHQLC() {

    Session hibSession = HibernateUtil.SFACTORY.openSession(); //open hibernate session factory
    Transaction txDB = null; //database transaction
    System.out.println("\n***** Listing employees using HQL criteria...\n");

    try {
        txDB = hibSession.beginTransaction(); //starts transaction
        // 1. Create a CriteriaBuilder instance by calling the Session.getCriteriaBuilder() method.
        CriteriaBuilder crbCritBuilder = hibSession.getCriteriaBuilder();
        // 2. Create a query object by creating an instance of the CriteriaQuery interface.
        CriteriaQuery<Employee> crqHQL = crbCritBuilder.createQuery(Employee.class);
        // 3. Set the query Root by calling the from() method on the CriteriaQuery
        // object to define a range variable in FROM clause.
        Root<Employee> rootEmployee = crqHQL.from(Employee.class);
        // 4. Specify what the type of the query result will be by calling the select()
        // method of the CriteriaQuery object
        crqHQL.select(rootEmployee).where(crbCritBuilder.gt(rootEmployee.get("dSalary"), 10000));
        crqHQL.orderBy(crbCritBuilder.desc(rootEmployee.get("dSalary")));
        // 5. Prepare the query for execution by creating a org.hibernate.query.Query
        // instance by calling the Session.createQuery() method, specifying the type of the query result.
        Query<Employee> qryHQL = hibSession.createQuery(crqHQL);
        // 6. Execute the query by calling the getResultList() or getSingleResult()
        // method on the org.hibernate.query.Query object.
        List<Employee> lstEmployee = qryHQL.getResultList();
        if (lstEmployee.isEmpty())
            System.out.println("***** No items found");
        else {
            for (Iterator<Employee> itEmployee = lstEmployee.iterator(); itEmployee.hasNext();) {
                Employee objEmployee = (Employee) itEmployee.next();
                System.out.print("First Name: " + objEmployee.getstFirstName() + " | ");
                System.out.print("Last Name: " + objEmployee.getstLastName() + " | ");
                System.out.println("Salary: " + objEmployee.getdSalary());
                Set<Certificate> relCertificates = objEmployee.getrelCertificates();
                for (Iterator<Certificate> itCertificate = relCertificates.iterator(); itCertificate.hasNext();) {
                    Certificate objCertificate = (Certificate) itCertificate.next();
                    System.out.println("Certificate: " + objCertificate.getstCertName());
                }
            }
            txDB.commit(); //ends transaction
        }
    } catch (HibernateException hibe) {
        if (txDB != null)
            txDB.rollback(); //something went wrong, so rollback
        hibe.printStackTrace();
    } finally {
        hibSession.close(); //close hibernate session
    }
}
```

We'll create a new method to list only the employees earning > 10000, order by salary desc



Upgrading to HQL Criteria. Main programme

```
public class TestHibernateMySQL {

    // MAIN PROGRAMME
    public static void main(String[] stArgs) {

        //Create new objects DAO for CRUD operations
        EmployeeDAO objEmployeeDAO = new EmployeeDAO();
        CertificateDAO objCertificateDAO = new CertificateDAO();

        //TRUNCATE TABLES. Delete all records from the tables
        objEmployeeDAO.deleteAllItems();
        objCertificateDAO.deleteAllItems();

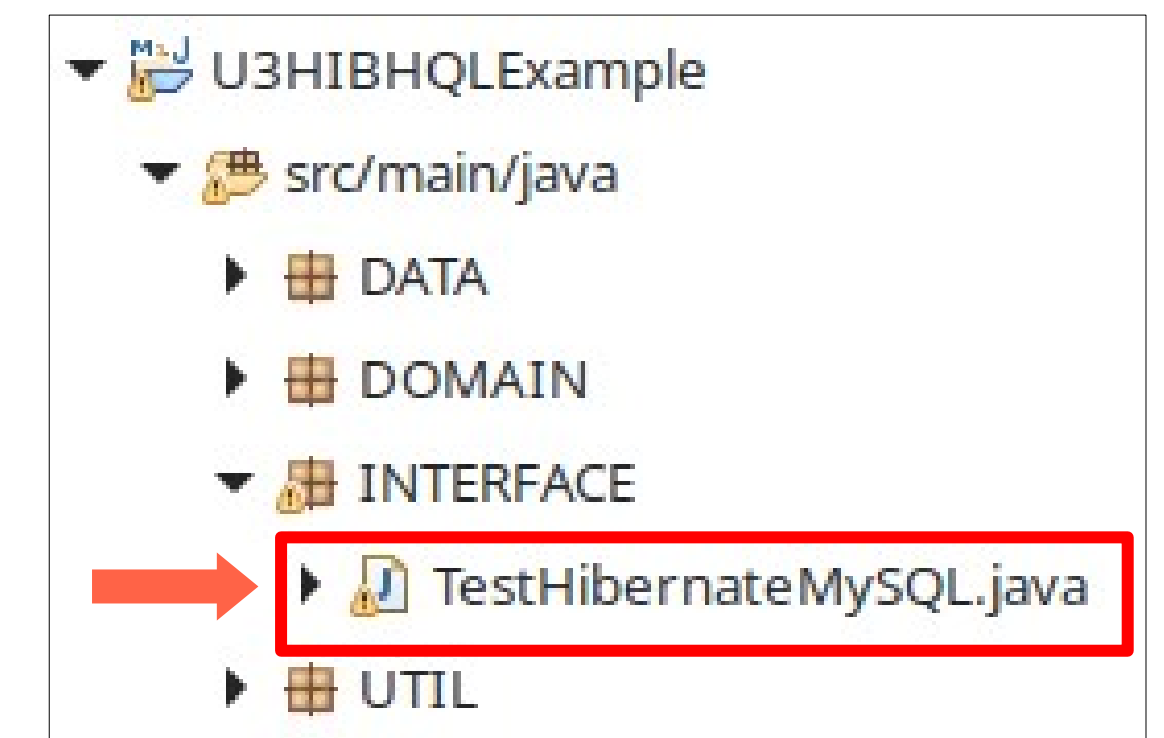
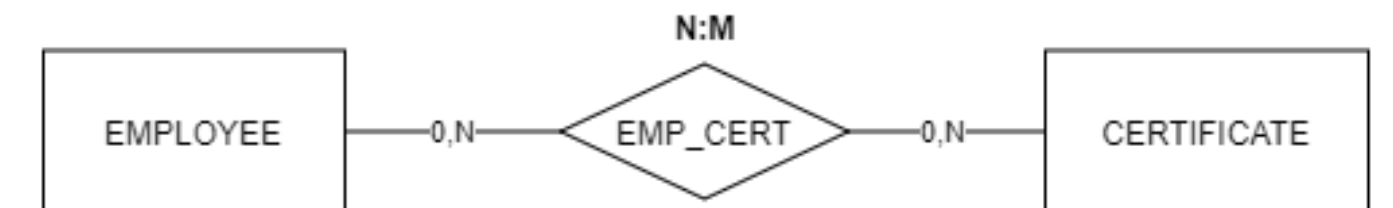
        /* Add records in the database */
        Certificate objCert1 = objCertificateDAO.addCertificate("MBA");
        Certificate objCert2 = objCertificateDAO.addCertificate("PMP");

        //Set of certificates
        HashSet<Certificate> hsetCertificates = new HashSet<Certificate>();
        hsetCertificates.add(objCert1);
        hsetCertificates.add(objCert2);

        /* Add records in the database */
        Employee objEmp1 = objEmployeeDAO.addEmployee("Alfred", "Vincent", 15000, hsetCertificates);
        Employee objEmp2 = objEmployeeDAO.addEmployee("John", "Gordon", 8000, hsetCertificates);
        Employee objEmp3 = objEmployeeDAO.addEmployee("Edgard", "Codd", 20000, hsetCertificates);
        Employee objEmp4 = objEmployeeDAO.addEmployee("Joseph", "Smith", 5000, hsetCertificates);
        Employee objEmp5 = objEmployeeDAO.addEmployee("Mary", "Jones", 15000, hsetCertificates);

        /* List down all the employees */
        //New way (HQL using criteria)
        objEmployeeDAO.listEmployeesHQLC();
        /* List down all the employees earning > 10000 */
        //New way (HQL using criteria)
        objEmployeeDAO.listRichEmployeesHQLC();

        //Close global hibernate session factory
        HibernateUtil.shutdownSessionFactory();
    }
}
```



5. ACTIVITIES FOR NEXT WEEK

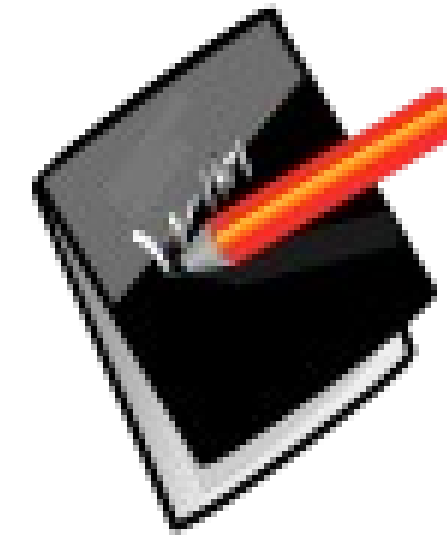
Proposed activities



Check the suggested exercises you will find at the “Aula Virtual”. **These activities are optional and non-assessable but** understanding these non-assessable activities is essential to solve the assessable task ahead.

Shortly you will find the proposed solutions.

6. BIBLIOGRAPHY



Resources

- Tutorialspoint. Hibernate tutorial. <https://www.tutorialspoint.com/hibernate/index.htm>
- Geekforgeeks. Hibernate – Query Language.
<https://www.geeksforgeeks.org/hibernate-query-language/>
- Baeldung. JPA Criteria Queries. <https://www.baeldung.com/hibernate-criteria-queries>
- Josep Cañellas Bornas, Isidre Guixà Miranda. Accés a dades. Desenvolupament d'aplicacions multiplataforma. Creative Commons. Departament d'Ensenyament, Institut Obert de Catalunya. Dipòsit legal: B. 29430-2013. <https://ioc.xtec.cat/educacio/recursos>
- Alberto Oliva Molina. Acceso a datos. UD 3. Herramientas de mapeo objeto relacional (ORM). IES Tubalcaín. Tarazona (Zaragoza, España).

