**Criteria**

* The **Criteria API** allows queries to be built at runtime without direct string manipulations.
* **Criteria** is an **API** from hibernate to write Queries in Object oriented manner rather than SQL or HQL.
* **Criteria** is also **database independent**, Because it internally generates HQL queries.
* We can execute only **SELECT** statements using Criteria; we can’t execute UPDATE, DELETE statements using 0.
* Criteria is suitable for executing **dynamic queries**
* Criteria API also include **Query by Example** (QBE) functionality for supplying example objects.
* Criteria also includes **projection** and **aggregation** methods

**NOTE**: Queries expressed as **criteria** are less readable than queries expressed in **HQL**.

To work with all criteria examples I am using the following Entity

**Account.java**

1. **package com.sekharit.hibernate.entity;**
2. **import javax.persistence.Column;**
3. **import javax.persistence.Entity;**
4. **import javax.persistence.GeneratedValue;**
5. **import javax.persistence.GenerationType;**
6. **import javax.persistence.Id;**
7. **import javax.persistence.Table;**
8. **import org.hibernate.annotations.GenericGenerator;**
9. **@Entity**
10. **@Table(name = "ACCOUNT")**
11. **public class Account {**
12. **@Id**
13. **@GenericGenerator(name = "myGen", strategy = "increment")**
14. **@GeneratedValue(strategy = GenerationType.AUTO, generator = "myGen")**
15. **@Column(name = "ACCNO")**
16. **private int accountId;**
17. **@Column(name = "NAME")**
18. **private String name;**
19. **@Column(name = "BALANCE")**
20. **private double balance;**
21. **public Account() {**
23. **}**
24. **public Account(int accoutnId, String name, double balance) {**
25. **This.accoutnId = accountId;**
26. **this.name = name;**
27. **this.balance = balance;**
28. **}**
29. **public int getAccountId() {**
30. **return accountId;**
31. **}**
32. **public void setAccountId(int accountId) {**
33. **this.accountId = accountId;**
34. **}**
35. **public String getName() {**
36. **return name;**
37. **}**
38. **public void setName(String name) {**
39. **this.name = name;**
40. **}**
41. **public double getBalance() {**
42. **return balance;**
43. **}**
44. **public void setBalance(double balance) {**
45. **this.balance = balance;**
46. **}**
47. **@Override**
48. **public String toString() {**
49. **return "Account [accountId=" + accountId + ", name=" + name**
50. **+ ", balance=" + balance + "]";**
51. **}**
52. **}**

Here’s a case study to retrieve a list of Account objects, with optional search criteria – balance, name, accountId, order by accountId.

**1. HQL example**

In HQL, you need to compare whether this is the first criteria to append the ‘where’ syntax. It’s work, but the long codes are ugly, cumbersome and error-prone string concatenation may cause security concern like SQL injection.

**Code block:**

1. **private static List<Account> getAccounts(Session session, Double balance,**
2. **String name, Integer accountId) {**
3. **boolean isFirst = true;**
4. **StringBuilder query = new StringBuilder("FROM Account a ");**
5. **if (balance != null) {**
6. **if (isFirst) {**
7. **query.append(" WHERE a.balance >= "+balance);**
8. **} else {**
9. **query.append(" AND a.balance >= "+balance);**
10. **}**
11. **isFirst = false;**
12. **}**
13. **if (name != null) {**
14. **if (isFirst) {**
15. **query.append(" WHERE a.name LIKE '"+name+"'");**
16. **} else {**
17. **query.append(" AND a.name LIKE '"+name+"'");**
18. **}**
19. **isFirst = false;**
20. **}**
21. **if (accountId != null) {**
22. **if (isFirst) {**
23. **query.append(" WHERE a.accountId >= "+accountId);**
24. **} else {**
25. **query.append(" AND a.accountId >= "+accountId);**
26. **}**
27. **isFirst = false;**
28. **}**
29. **query.append(" ORDER BY a.accountId");**
30. **Query result = session.createQuery(query.toString());**
31. **List<Account> accounts = result.list();**
32. **sop(accounts.size());**
33. **return accounts;**
34. **}**

**2. Criteria example**

In Criteria, you do not need to compare whether this is the first criteria to append the ‘where’ syntax.. The line of code is reduce and everything is handled in a more elegant and object oriented way.

**Code block:**

1. **private static List<Account> getAccounts(Session session, Double balance,**
2. **String name, Integer accountId) {**
3. **boolean isFirst = true;**
4. **Criteria criteria = session.createCriteria(Account.class);**
5. **if (balance != null) {**
6. **criteria.add(Restrictions.ge("balance", balance));**
7. **}**
8. **if (name != null) {**
9. **criteria.add(Restrictions.like("name", name));**
10. **}**
11. **if (accountId != null) {**
12. **criteria.add(Restrictions.ge("accountId", accountId));**
13. **}**
14. **criteria.addOrder(Order.asc("accountId"));**
15. **List<Account> accounts = criteria.list();**
16. **sop(accounts);**
17. **return accounts;**
18. **}**

**Steps to work with Criteria API**

**Step 1**: Create **org.hibernate.Criteria** Object

**Criteria criteria = session.createCriteria(EntityClassName.class);**

**Step 2**: Create **org.hibernate.criterion.Criterion** Object per each

condition of the query and add to **Criteria** object.

**Criterion criterion = Restrictions.eq(“propertyName”, propertyValue);**

**criteria.add(criterion);**

**Setp 3**: Execute **org.hibernate.Criteria** object (by calling **list()** method on **Criteria** object)

**List list = criteria.list();**

**Q) Write and execute Criteria API code for the following SQL statement?**

**SQL:** SELECT \* FROM ACCOUNT WHERE NAME=’sekhar’ AND BALANCE>1800;

**CRITERIA:**

1. **// SELECT \* FROM ACCOUNT**
2. **Criteria criteria = session.createCriteria(Account.class);**
3. **// NAME=’sekhar’**
4. **Criterion nameCriterion = Restrictions.eq("name", "sekhar");**
5. **// BALANCE>1800**
6. **Criterion balanceCriterion = Restrictions.gt("balance", 1800.0);**
7. **// NAME=’sekhar’ AND BALANCE>1800**
8. **Criterion criterion = Restrictions.and(nameCriterion, balanceCriterion);**
9. **// SELECT \* FROM ACCOUNT WHERE NAME=’sekhar’ AND BALANCE>1800;**
10. **criteria.add(criterion);**
11. **List<Account> accounts = criteria.list();**
12. **for (Account account : accounts) {**
13. **sop(account);**
14. **}**

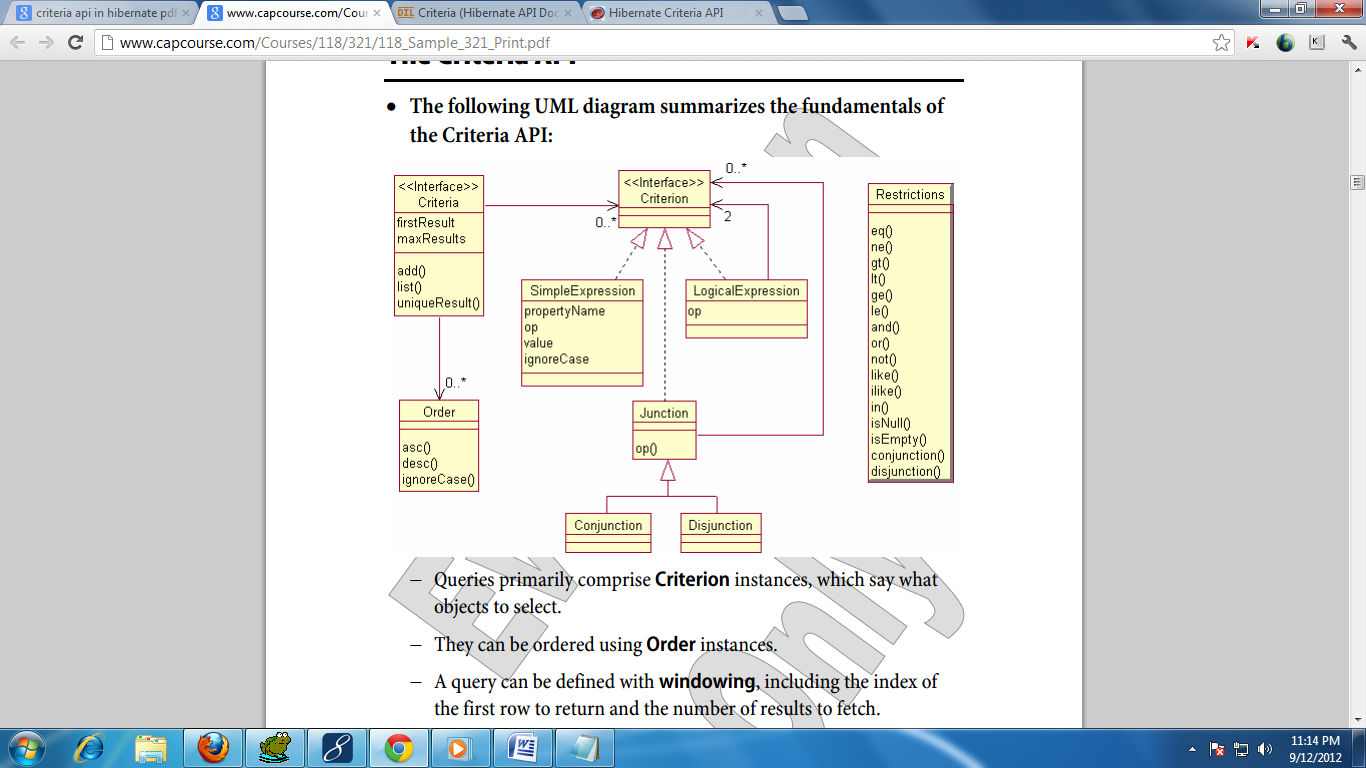
**Q) How to implement the previous code using method chaining concept?**

**NOTE:** As most Hibernate classes, the methods in Criteria return the **this** reference, so additional calls can be chained.

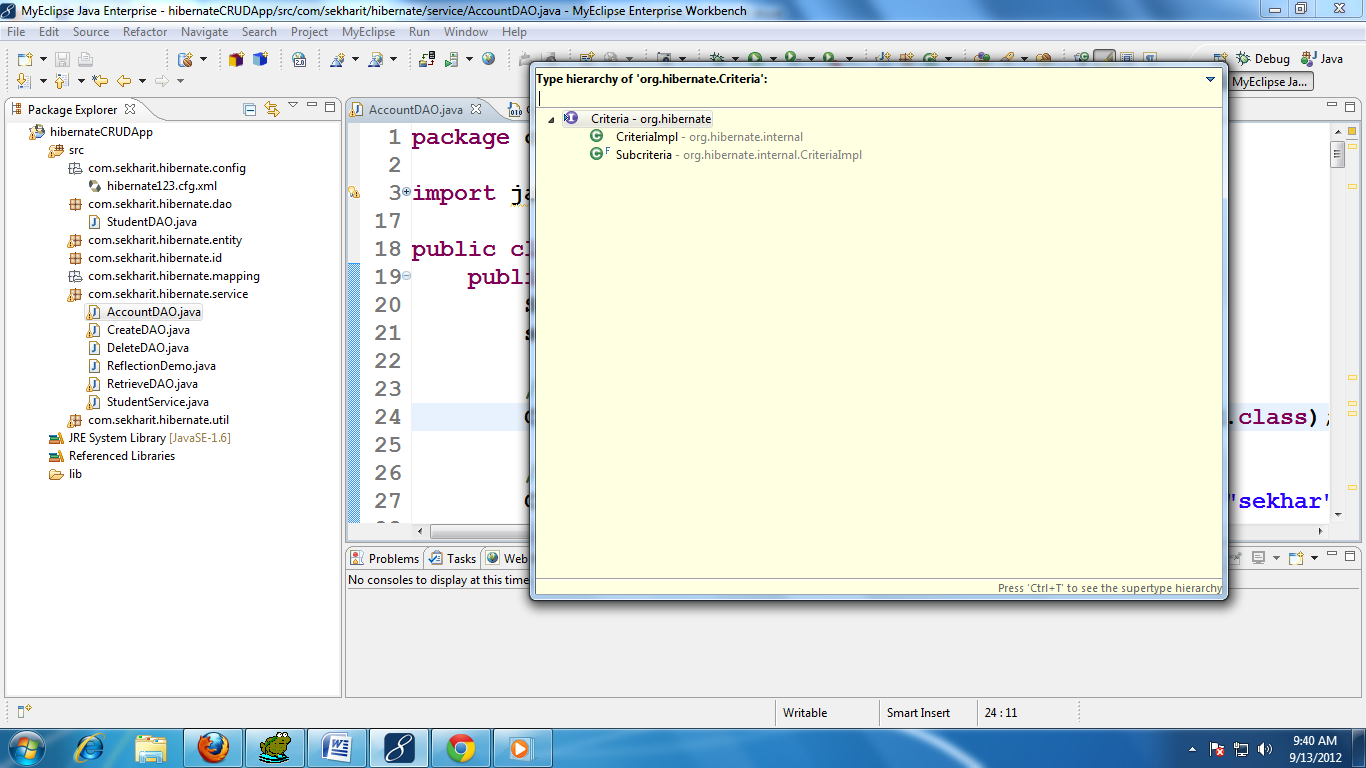
1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.eq("name", "sekhar"))**
3. **.add(Restrictions.gt("balance", 1800.0))**
4. **.list();**
5. **for (Account account : accounts) {**
6. **sop(account);**
7. **}**

**Q)** What do you know about **org.hibernate.Criteria**, **org.hibernate.criterion.Criterion**, **org.hibernate.criterion.Restrictions** and **org.hibernate.criterion.Expression**?

* The following UML diagram summarizes the fundamentals of the Criteria API:

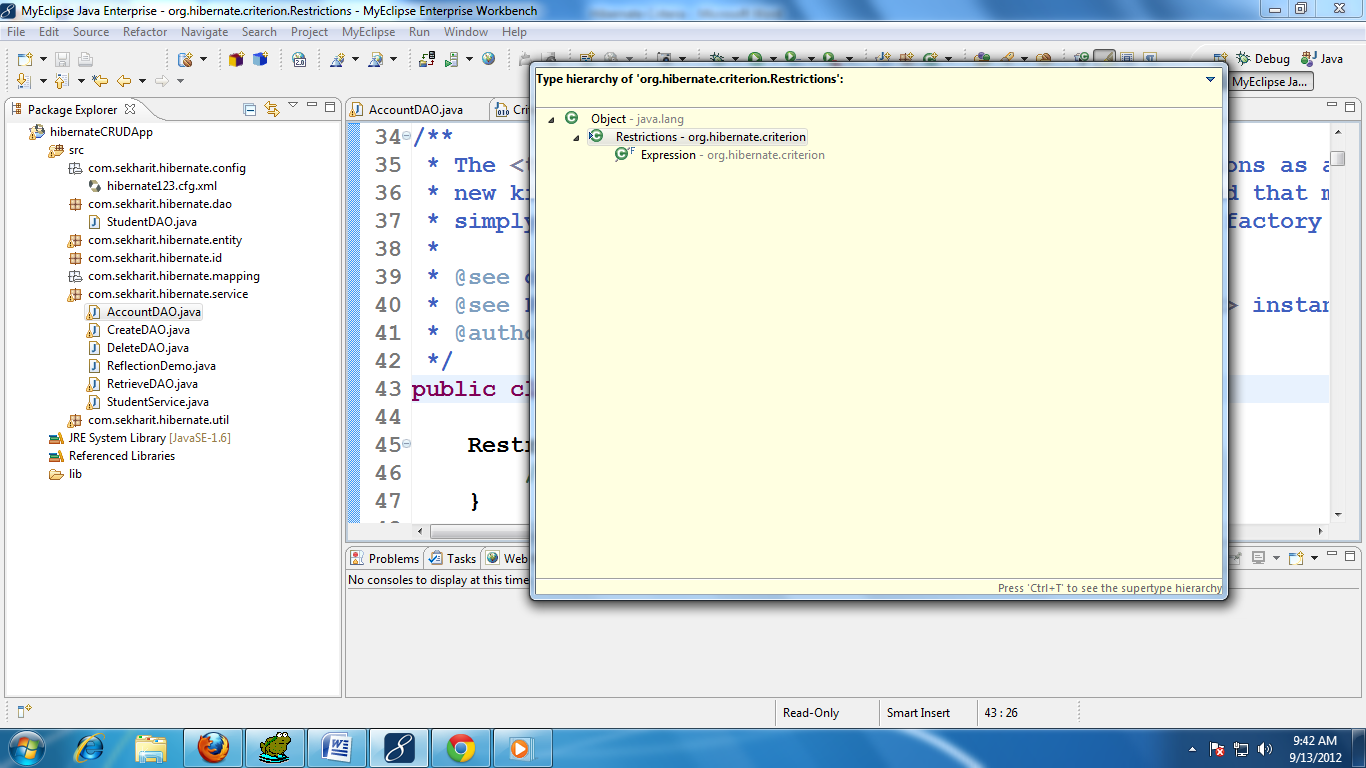


* **Criteria** is aninterface used to represent query in Object oriented format. It contains methods to add **Criterion** objects, **Order** objects, **Projection** objects, **pagination** methods …etc. **CriteriaImpl** is the implementation class of **Criteria**.



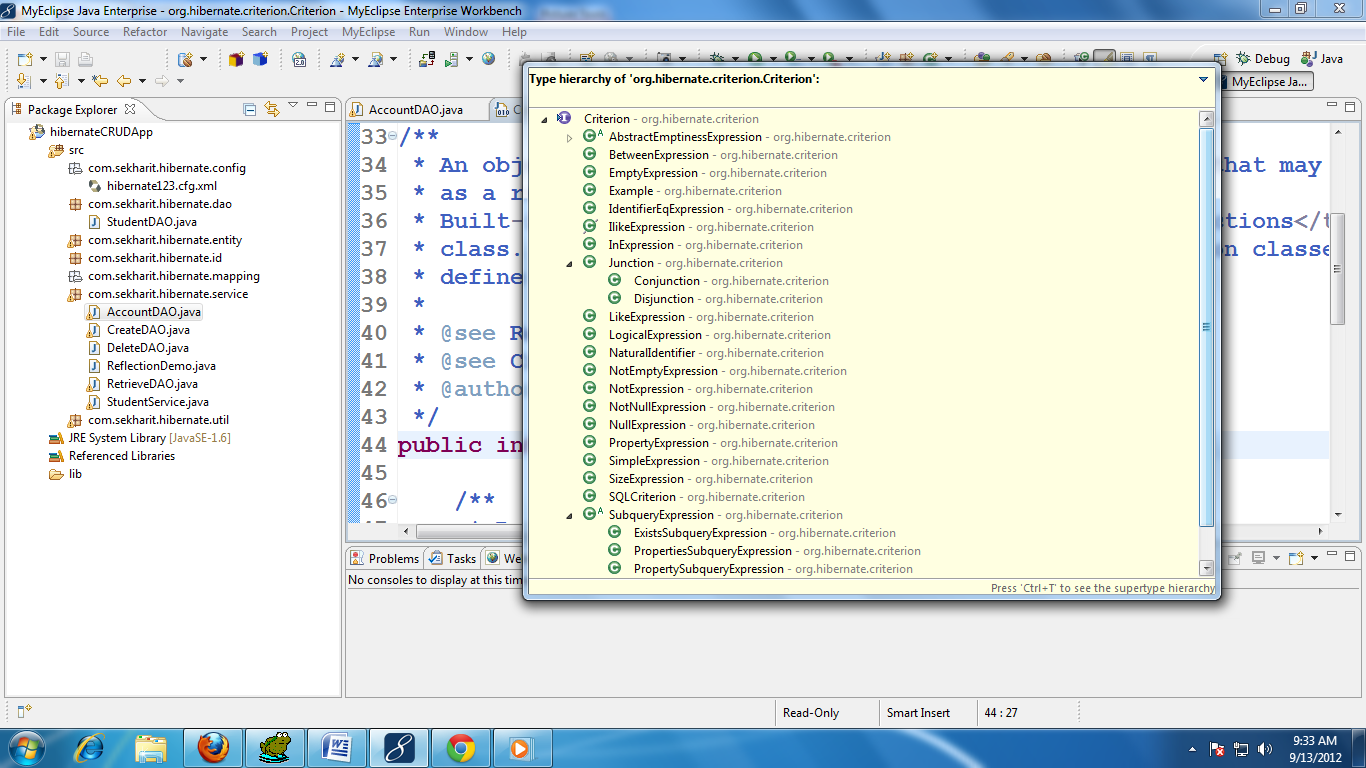
* **Restrictions** is class which has methods to define conditions. These class methods return different **XXXExpression** classes which implements **Criterion** interface. So **Restrictions** is a static factory for **Criterion** instances.

* **Expression** class is child class of **Restrictions.** It defines methods to implement native SQL. As latest versions of Hibernate provides separate **Native SQL** API, this class made as deprecated.

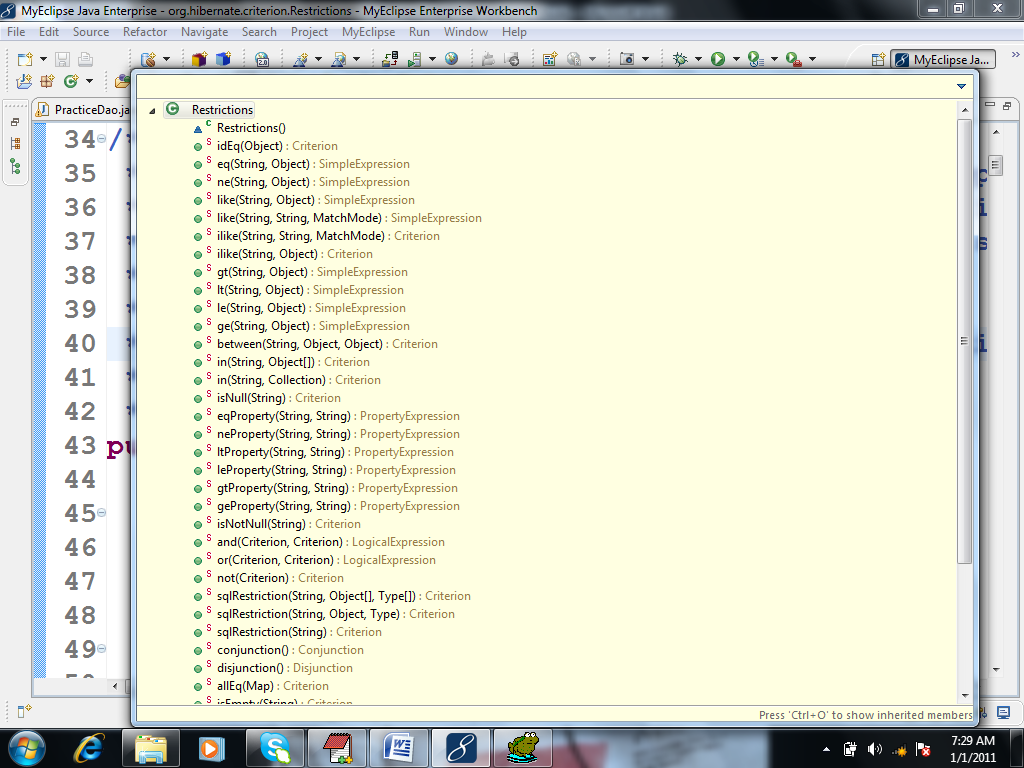


* **Criterion** is an interface which is used to represent one condition in object oriented manner. By calling methods on **Restrictions** we will get this object. There are different types of **XXXExpression** classes which are implementing **Criterion** interface. But all those implementation are Abstracted with **Criterion** interface

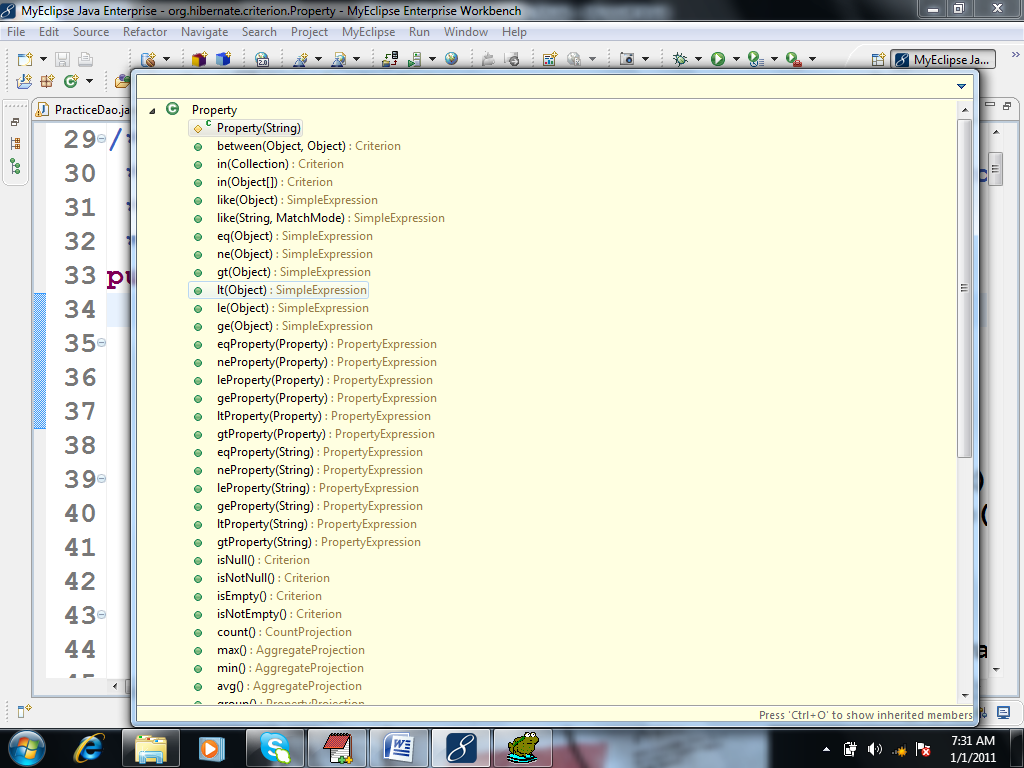
|  |  |  |
| --- | --- | --- |
| **Expression class** | **Restrictions method** | **SQL operator/function** |
| SimpleExpression | eq(prop, val) | = |
| SimpleExpression | ne(prop, val) | !=, <> |
| SimpleExpression | like(prop, val) [**case-sensitive]** | LIKE |
| SimpleExpression | gt(prop, val) | > |
| SimpleExpression | lt (prop, val) | < |
| SimpleExpression | ge(prop, val) | >= |
| SimpleExpression | le(prop, val) | <= |
| IdentifierExpression | idEq(val) | <Id-column> = |
| ILikeExpression | ilike(prop, val) [**case-in-sensitive]** | LIKE |
| BetweenExptession | between(prop, val, val) | BETWEEN <lowerLmit> AND <upperLimit> |
| InExpression | in(prop, val[]) | IN(val1, val2….) |
| NullExpression | isNull(prop) | <column> IS NULL |
| NullExpression | isNotNull(prop) | <column> IS NOT NULL |
| LogicalExpression | and (criterion1, criterion2) | <expr1> AND <expr2> |
| LogicalExpression | or(criterion1, criterion2) | <expr1> OR <expr2> |
| NotExpression | not(criterioin) | ! |
| EmptyExpression | isEmpty(prop) | EMPTY |



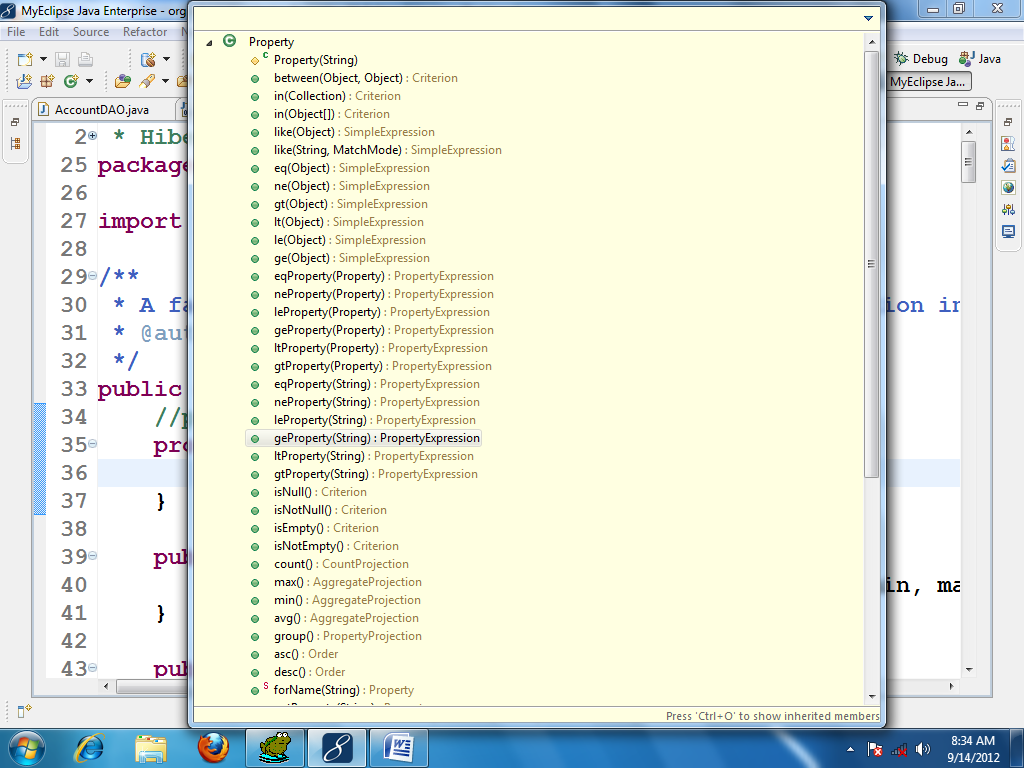
* **org.hibernate.criterion.Property** is a class available in hibernate API.
* **Property** is used to create **Criterion** objects(means creating conditions of the query).
* **Property** is just like **Restrictions** class. Both are meant for creating **Criterion** objects.
* If we are using same property of entity, on multiple conditions it is better to go for **Property** rather than **Restrictions**.
* You can create a **Property** by calling **Property.forName(“property-Name”)**
* If we want to compare properties of entity, we can use the following methods of **Restrictions** class.



If we want to compare properties we can use the following methods of **Property** class.



**org.hibernate.criterion.Property** class methods are…



Q) Write and execute Criteria API code for the following SQL statement? Use **org.hibernate.criterion.Property** class to create conditions

**SQL:** SELECT \* FROM ACCOUNT WHERE NAME=’sekhar’ AND BALANCE>1800;

**CRITERIA:**

1. **// SELECT \* FROM ACCOUNT**
2. **Criteria criteria = session.createCriteria(Account.class);**
3. **// NAME=’sekhar’**
4. **Property nameProperty = Property.forName("name");**
5. **Criterion nameCriterion = nameProperty.eq("sekhar");**
6. **// BALANCE>1800**
7. **Property balanceProperty = Property.forName("balance");**
8. **Criterion balanceCriterion = balanceProperty.gt(1800.0);**
9. **// NAME=’sekhar’ AND BALANCE>1800;**
10. **Criterion criterion = Restrictions.and(nameCriterion, balanceCriterion);**
11. **// SELECT \* FROM ACCOUNT WHERE NAME=’sekhar’ AND BALANCE>1800;**
12. **criteria.add(criterion);**
13. **List<Account> accounts = criteria.list();**
14. **for (Account account : accounts) {**
15. **sop(account);**
16. **}**

OR

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(**
3. **Restrictions.and(**
4. **Property.forName("name").eq("sekhar"),**
5. **Property.forName("balance").gt(1800.0)))**
6. **.list();**

**Q.) How to apply Pagination in criteria ?**

1. **Criteria criteria = session.createCriteria(Account.class);**
2. **criteria.setFirstResult(5);**
3. **criteria.setMaxResults(5);**
4. **List<Account> accounts = criteria.list();**
5. **for (Account account : accounts) {**
6. **sop(account);**
7. **}**

**Examples based on different conditions**

**Example:** Id Equal Restriction

**SQL:** SELECT \* FROM ACCOUNT WHERE **ACCNO=5001**

**CRITERIA:**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.eq("accountId", 5001))**
3. **.list();**

**OR**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.idEq(5001))**
3. **.list();**

**OR**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Property.forName("accountId").eq(5001))**
3. **.list();**

## Example: Sorting the result

**SQL:** SELECT \* FROM ACCOUNT **ORDER BY** BALANCE **DESC**, NAME **ASC**, ACCNO **DESC**

**CRITERIA:**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.addOrder(Property.forName("balance").desc())**
3. **.addOrder(Property.forName("name").asc())**
4. **.addOrder(Property.forName("accountId").desc())**
5. **.list();**

OR

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.addOrder(Order.desc("balance"))**
3. **.addOrder(Order.asc("name"))**
4. **.addOrder(Order.desc("accountId"))**
5. **.list();**

**Example:** Logical operators Restriction

**SQL:** SELECT \* FROM ACCOUNT WHERE ACCNO **>=** 5001 **AND** ACCNO **<=** 5005

**CRITERIA:**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.or(**
3. **Restrictions.ge("accountId", 5001),**
4. **Restrictions.le("accountId", 5005)))**
5. **.list();**

**SQL:** SELECT \* FROM ACCOUNT WHERE ACCNO **>** 5001 **OR** ACCNO **<** 5005

**CRITERIA:**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.and(**
3. **Restrictions.gt("accountId", 5001),**
4. **Restrictions.lt("accountId", 5005)))**
5. **.list();**

OR

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add( Restrictions.gt("accountId", 5001))**
3. **.add( Restrictions.lt("accountId", 5005))**
4. **.list();**

**NOTE:**  If we directly add multiple **criterion** objects to **Criteria**, by default “**AND**” logical operator will be applied among the conditions of the generated query.

**Example:** Conjunction and Disjunction

Conditions can be combined using Restrictions.disjunction() and Restrictions.conjunction().

− No arguments are passed to either method.

− Criterion instances are then added to the conjunction or disjunction using their respective add methods.

− The conjunction or disjunction is then added to a Criteria instance and evaluated.

**SQL:** SELECT \* FROM ACCOUNT WHERE ACCNO>5001 **AND** NAME =’sekhar’ **AND** BALANCE>2500.0 **AND** BALANCE<9200.0

**CRITERIA:**

1. **Criteria criteria = session.createCriteria(Account.class);**
3. **Criterion c1 = Restrictions.gt("accountId", 5001);**
4. **Criterion c2 = Restrictions.eq("name", "sekhar");**
5. **Criterion c3 = Restrictions.gt("balance", 2500.0);**
6. **Criterion c4 = Restrictions.lt("balance", 9200.0);**
8. **Criterion c5 = Restrictions.and(c1,c2);**
9. **Criterion c6 = Restrictions.and(c3,c4);**
10. **Criterion c7 = Restrictions.and(c5,c6);**
12. **criteria.add(c7);**
13. **List<Account> accounts = criteria.list();**

**OR**

1. **Criteria criteria = session.createCriteria(Account.class);**
3. **Criterion c1 = Restrictions.gt("accountId", 5001);**
4. **Criterion c2 = Restrictions.eq("name", "sekhar");**
5. **Criterion c3 = Restrictions.gt("balance", 2500.0);**
6. **Criterion c4 = Restrictions.lt("balance", 9200.0);**
8. **Conjunction conjunction = Restrictions.conjunction();**
9. **conjunction.add(c1);**
10. **conjunction.add(c2);**
11. **conjunction.add(c3);**
12. **conjunction.add(c4);**
14. **criteria.add(conjunction);**
15. **List<Account> accounts = criteria.list();**

**OR**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.conjunction()**
3. **.add(Restrictions.gt("accountId", 5001))**
4. **.add(Restrictions.eq("name", "sekhar"))**
5. **.add(Restrictions.gt("balance", 2500.0))**
6. **.add(Restrictions.lt("balance", 9200.0)))**
7. **.list();**

**SQL:** SELECT \* FROM ACCOUNT WHERE ACCNO>5001 **OR** NAME =’sekhar’ **OR** BALANCE>2500.0 **OR** BALANCE<9200.0

**CRITERIA:**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.disjunction()**
3. **.add(Restrictions.gt("accountId", 5001))**
4. **.add(Restrictions.eq("name", "sekhar"))**
5. **.add(Restrictions.gt("balance", 2500.0))**
6. **.add(Restrictions.lt("balance", 9200.0)))**
7. **.list();**

**Example:** Equal and Not-Equal Restriction

**SQL:** SELECT \* FROM ACCOUNT WHERE NAME <> ‘somu’ OR NAME = ‘sekhar’

**CRITERIA:**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.or(**
3. **Restrictions.ne("name", "somu"),**
4. **Restrictions.eq("name", "sekhar")))**
5. **.list();**

OR

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.or(**
3. **Property.forName("name").ne("somu"),**
4. **Property.forName("name").eq("sekhar")))**
5. **.list();**

**Example:** Like Restriction (case-sensitive)

**SQL:** SELECT \* FROM ACCOUNT WHERE NAME LIKE ‘%sekhar%’ OR NAME LIKE ‘cherry%’

**CRITERIA:**

The **like** operator allows wildcard searches, where the wildcard symbols are **%** and **\_**, just as in SQL:   
  
For criteria queries, wildcard searches may use either the same wildcard symbols or specify a **MatchMode**. Hibernate provides the **MatchMode** as part of the Criteria query API; we use it for writing string match expressions without string manipulation. The allowed **MatchModes** are **START**, **END**, **ANYWHERE**, and **EXACT**.

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.or(**
3. **Restrictions.like("name", "%sekhar%"),**
4. **Restrictions.like("name", "cherry%")))**
5. **.list();**

**OR**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.or(**
3. **Restrictions.like("name", "sekhar", MatchMode.ANYWHERE),**
4. **Restrictions.like("name", "cherry", MatchMode.START)))**
5. **.list();**

**OR**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.or(**
3. **Restrictions.like("name", "SEKHAR", MatchMode.ANYWHERE),**
4. **Restrictions.like("name", "CHERRY", MatchMode.START)))**
5. **.list();**

**OR**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.or(**
3. **Property.forName("name").like("%sekhar%"),**
4. **Property.forName("name").like("cherry%")))**
5. **.list();**

**OR**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.or(**
3. **Property.forName("name").like("sekhar", MatchMode.ANYWHERE),**
4. **Property.forName("name").like("cherry", MatchMode.START)))**
5. **.list();**

**Example:** Like Restriction (case-insensitive)

**SQL:** SELECT \* FROM ACCOUNT WHERE NAME LIKE ‘%sekhar%’ AND NAME LIKE ‘%somu%’

**CRITERIA:** Re-write the above code with ilike() method instead like() method

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.or(**
3. **Restrictions.ilike("name", "%sekhar%"),**
4. **Restrictions.ilike("name", "cherry%")))**
5. **.list();**

**Example:** is Null, is Not Null

**SQL:** SELECT \* FROM ACCOUNT WHERE NAME IS NULL and BALANCE IS NOT NULL

**CRITERIA:**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.isNull("name"))**
3. **.add(Restrictions.isNotNull("balance"))**
4. **.list();**

**OR**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Property.forName("name").isNull())**
3. **.add(Property.forName("balance").isNotNull())**
4. **.list();**

**Example:** When to use Propery class in general?

**SQL:** SELECT \* FROM ACCOUNT WHERE NAME IS NOT NULL AND NAME LIKE '%sekahr%' OR NAME LIKE '%somu%'

**CRITERIA:** In the query there are multiple conditions on the same column.

1. **List<Account> accounts =**
2. **session.createCriteria(Account.class)**
3. **.add(Restrictions.and(**
4. **Restrictions.isNotNull("name"),**
5. **Restrictions.or(**
6. **Restrictions.like("name", "sekhar", MatchMode.ANYWHERE),**
7. **Restrictions.like("name", "somu",MatchMode.ANYWHERE))))**
8. **.list();**

**OR**

1. **List<Account> accounts =**
2. **session.createCriteria(Account.class)**
3. **.add(Restrictions.and(**
4. **Property.forName("name").isNotNull(),**
5. **Restrictions.or(**
6. **Property.forName("name").like( "sekhar", MatchMode.ANYWHERE),**
7. **Property.forName("name").like("somu",MatchMode.ANYWHERE))))**
8. **.list();**

**OR**

1. **Property nameProperty = Property.forName("name");**
2. **List<Account> accounts = session.createCriteria(Account.class)**
3. **.add(Restrictions.and(**
4. **nameProperty.isNotNull(),**
5. **Restrictions.or(**
6. **nameProperty.like( "sekhar", MatchMode.ANYWHERE),**
7. **nameProperty.like("somu",MatchMode.ANYWHERE))))**
8. **.list();**

**NOTE:** In this the last code block we are creating **nameProperty** only once, and we are reusing the same in multiple conditions, this is the exact scenario, where we need to use Property class to write conditions

**Example:** Between Restriction

**SQL:** SELECT \* FROM ACCOUNT WHERE BALANCE BETWEN 4560.0 AND 8689.0

**CRITERIA:**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.between("balance", 4560.0, 8689.0))**
3. **.list();**

OR

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Property.forName("balance").between( 4560.0, 8689.0))**
3. **.list();**

**Example:** In Restriction

**SQL:** SELECT \* FROM ACCOUNT WHERE ACCNO IN(5001,5009,5005,5004)

**CRITERIA:**

1. **Object[] valuesArray = new Object[]{5001,5009,5005,5004};**
2. **List<Account> accounts = session.createCriteria(Account.class)**
3. **.add(Restrictions.in("accountId", valuesArray))**
4. **.list();**

**OR**

1. **List<Integer> valuesList = new ArrayList<Integer>();**
2. **valuesList.add(5001);**
3. **valuesList.add(5009);**
4. **valuesList.add(5005);**
5. **valuesList.add(5004);**
6. **List<Account> accounts = session.createCriteria(Account.class)**
7. **.add(Restrictions.in("accountId", valuesList))**
8. **.list();**

**Example:** Property comparison

**SQL:** SELECT \* FROM ACCOUNT WHERE ACCNO < BALANCE OR NAME LIKE ‘%sekhar%

**CRITERIA:**

1. **Criteria criteria = session.createCriteria(Account.class);**
2. **Criterion criterion1 = Restrictions.ltProperty("accno", "balance");**
3. **Criterion criterion2 = Restrictions.like("name", "sekhar", MatchMode.ANYWHERE);**
4. **Criterion criterion3 = Restrictions.or(criterion1, criterion2);**
5. **criteria.add(criterion3);**
6. **List<Account> accounts = criteria.list();**

OR

1. **Property accnoProperty = Property.forName("accno");**
2. **Property balanceProperty = Property.forName("balance");**
3. **Property nameProperty = Property.forName("name");**
4. **Criteria criteria = session.createCriteria(Account.class);**
5. **Criterion criterion1 = accnoProperty.ltProperty(balanceProperty);**
6. **Criterion criterion2 = nameProperty.like("sekhar", MatchMode.ANYWHERE);**
7. **Criterion criterion3 = Restrictions.or(criterion1, criterion2);**
8. **criteria.add(criterion3);**
9. **List<Account> accounts = criteria.list();**

**Example:** Equal All

**SQL:** SELECT \* FROM ACCOUNT WHERE ACCNO = 5001 AND BALANCE =4500.0 AND NAME = ‘sekhar’

**CRITERIA:**

1. **Map<String, Object> map = new HashMap<String, Object>();**
2. **map.put("accountId", 5001);**
3. **map.put("name", "sekhar");**
4. **map.put("balance", 4500.0);**
6. **List<Account> accounts = session.createCriteria(Account.class)**
7. **.add(Restrictions.allEq(map))**
8. **.list();**

**Example:** SQL in Criteria

**SQL:** SELECT \* FROM ACCOUNT WHERE ACCNO = 5001

**CRITERIA:**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.sqlRestriction("{alias}.ACCNO=?",**
3. **5001, StandardBasicTypes.INTEGER))**
4. **.list();**

* To write SQL syntax in criteria we use **sqlRestriction**() method of Restrictions class.
* **{alias}** is because Hibernate internally generates alias name. So we should use **{alias}** before the column names.
* Writing SQL syntax with Criteria is not at all advisable. Because it is database dependent.

**NOTE**: When we are unable to implement conditions using criteria API, then only we will go for **sqlRestrictions()**.

**NOTE:** In the previous version of Hibernate to represent types we used to use **~~org.hibernate.Hibernate~~** class. But this class **deprecated** in the latest versions, So to represent types in the latest versions we can use **org.hibernate.type.StandardBasicTypes**

**Example:**

**SQL:** SELECT \* FROM ACCOUNT

WHERE

( ACCNO BETWEEN 5001 AND 5004 )

OR

(NAME LIKE ‘%sekhar%’)

**CRITERIA:**

1. **Criteria criteria = session.createCriteria(Account.class);**
2. **Criterion criterion1 = Restrictions.between("accountId", 5001, 5004);**
3. **Criterion criterion2 = Restrictions.like("name", "%sekhar%");**
4. **Criterion criterion3 = Restrictions.and(criterion1, criterion2);**
5. **criteria.add(criterion3);**
6. **List accounts = criteria.list();**

**(OR)**

1. **List accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.and(**
3. **Restrictions.between("accno", 1001, 1004),**
4. **Restrictions.like("name", "%a%")))**
5. **.list();**

**Example:**

**SQL:**SELECT \* FROM ACCOUNT

WHERE

NAME IN (‘sekhar’, ‘kesavareddy’)

AND (

ACCNO IS NULL

OR ACCNO=5001

OR ACCNO=5002

OR ACCNO=5003

)

**CRITERIA:**

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.in("name", new Object[]{"sekhar","kesavareddy"}))**
3. **.add(Restrictions.disjunction()**
4. **.add(Restrictions.isNull("accountId"))**
5. **.add(Restrictions.eq("accountId ",** 5001**))**
6. **.add(Restrictions.eq("accountId ",** 5002**))**
7. **.add(Restrictions.eq("accountId ",** 5003**))**
8. **).list();**

OR

1. **Property accountId = Property.forName("accountId ");**
2. **List accounts = sess.createCriteria(Account.class)**
3. **.add( Restrictions.disjunction()**
4. **.add(accountId.isNull() )**
5. **.add(accountId.eq(**5001**) )**
6. **.add(accountId.eq(**5002**) )**
7. **.add(accountId.eq(**5003**) ))**
8. **.add( Property.forName("name").in( new String[] { "sekhar", "kesavareddy" } ))**
9. **.list();**

**Example:**

**SQL:** SELECT \* FROM

( SELECT \* FROM ACCOUNT

WHERE

NAME LIKE ‘sekhar’

ORDER BY

NAME ASC,

ACCNO DESC

)

WHERE

ROWNUM <= 5

**CRITERIA:**

1. **Criteria criteria = session.createCriteria(Account.class);**
2. **Criterion c1 = Restrictions.like("name", "sekhar");**
3. **Order nameOrder = Order.asc("name");**
4. **Order accnoOrder = Order.desc("accountId");**
6. **criteria.add(c1);**
7. **criteria.addOrder(nameOrder);**
8. **criteria.addOrder(accnoOrder);**
10. **criteria.setMaxResults(5);**
12. **List<Account> accounts = criteria.list();**

OR

1. **List<Account> accounts = session.createCriteria(Account.class)**
2. **.add(Restrictions.like("name", "sekhar"))**
3. **.addOrder(Order.asc("name"))**
4. **.addOrder(Order.desc("accno"))**
5. **.setMaxResults(10)**
6. **.list();**

**Example:** **uniqueResult()** method

**SQL:** SELECT \* FROM ACCNO =5001

**CRITERIA:**

1. **Account account = (Account) session.createCriteria(Account.class)**
2. **.add(Restrictions.idEq(5001))**
3. **.uniqueResult();**
4. **sop(account);**

**Query By Example**

* Query By **org.hibernate.criterion.Example** provides another style of searching.
* Rather than build a query by programmatically adding conditions to a Criteria object, partially populate an instance of the desired object.

− The partially populated instance is an example.

− Hibernate then builds the Criteria query from the example.

* To convert an object into an example, use the **create()** method in the **org.hibernate.criterion.Example** class.

**public static Example create(Object entity);**

* The trick with QBE is that when the query is translated into SQL, all properties of the example that are not null are used in the query.

− To drop a property from the list, use the **excludeProperty()**method.

− For numerical properties, use the **excludeZeros()** method.

* **Example** also has the **ignoreCase()** method, which does what it sounds like, and the **enableLike()** method, which is used for string comparisons.

**Example:**

**SQL:** SELECT \* FROM ACCOUNT WHERE NAME=’sekhar’ AND BALANCE = 4200.0

**CRITERIA:**

1. **Account account = new Account();**
2. **account.setName("sekhar");**
3. **account.setBalance(4200.0);**
5. **Example example = Example.create(account);**
6. **Criteria criteria = session.createCriteria(Account.class);**
7. **criteria.add(example);**
8. **List<Account> accounts = criteria.list();**

OR

1. **Account account = new Account();**
2. **account.setName("sekhar");**
3. **account.setBalance(4200.0);**
5. **Example example = Example.create(account)**
6. **.excludeZeroes()//exclude zero valued properties**
7. **.enableLike(MatchMode.START) //use like for string comparisons**
8. **.excludeProperty("accountId") //exclude the property named "accountId"**
9. **.ignoreCase(); //perform case insensitive string comparisons**
11. **Criteria criteria = session.createCriteria(Account.class);**
12. **criteria.add(example);**
13. **List<Account> accounts = criteria.list();**

**Projections**

* Each of the above examples we selected all properties of the Entity. Instead if we want to select particular properties (columns) or to perform aggregate functions we use Projections.
* We have done this in **HQL** also, but there we don’t need any API support. As part of **HQL** query itself we can specify required properties (columns) names. But in **Criteria** we have separate API to specify required properties. i.e **Projection** API.

**Q) What are the steps work with Projections in Criteria ?**

Step1: Create **Projection** objects

**Projection p1= Projections.property(…);**

Step2: create **ProjectionList** object

**ProjectionList pList = Projections.projectionList();**

Step3: add projection objects to pList;

**pList.add(p1);**

Step 4: set pList to criteria object

**criteria.setProjection(pList);**

The following UML diagram summarizes the fundamentals of the Projections API:



* **Projection** is an interface, which represents one selected property or selected aggregate function.
* **ProjectionList** is a class, which is used to hold multiple **Projection** objects.
* **Projections** is a Util class, which contains methods to create **Projection** objects. These methods return different **XXXPrjection** objects.
* But when we call methods on **Projections** we can assign those return types to **Projection** interface. Because all **XXXPrjection** classes directly or indirectly implements **Projection** interface.

|  |  |  |
| --- | --- | --- |
| **Projection class** | **Method** | **Value/SQL function** |
| Distinct | distinct(Projection) | DISTINCT(COLUMN) |
| RowCountProjection | rowCount() | COUNT(\*) |
| CountProjection | count(property) | COUNT(COLUMN) |
| CountProjection | countDistinct(property) | COUNT(DISTINCT COLUMN) |
| AggregateProjection | max(property) | MAX(COLUMN) |
| AggregateProjection | min(property) | MIN(COLUMN) |
| AvgProjection | avg(property) | AVG(COLUMN) |
| AggregateProjection | sum(property) | SUM(COLUMN) |
| PropertyProjection | groupProperty(property) | GROUP BY <COLUMN> |
| IdentifierProjection | id () | <ID-COLUMN> |
| SQLProjection | sqlProjection(String) | SQL QUERY |

**Examples on Projections**

**Example**:

**SQL**: SELECT NAME, BALANCE FROM ACCOUNT

**CRITERIA**:

1. **Criteria criteria = session.createCriteria(Account.class);**
2. **ProjectionList projectionList = Projections.projectionList();**
3. **Projection projection1 = Projections.property("name");**
4. **Projection projection2 = Projections.property("balance");**
6. **projectionList.add(projection1);**
7. **projectionList.add(projection2);**
9. **criteria.setProjection(projectionList);**
10. **List<?> list = criteria.list();**

**(OR)**

1. **List<?> list = session.createCriteria(Account.class)**
2. **.setProjection(**
3. **Projections.projectionList()**
4. **.add(Projections.property("name"))**
5. **.add(Projections.property("balance")))**
6. **.list();**

**(OR)**

1. **List<?> list= session.createCriteria(Account.class)**
2. **.setProjection(**
3. **Projections.projectionList()**
4. **.add(Property.forName("name"))**
5. **.add(Property.forName("balance")))**
6. **.list();**

**NOTE**: if only one selected property is there then we can set **Projection** object directly to **Criteria**, But if we have multiple selected properties, Then we should go for **ProjectionList**.

**SQL**: SELECT NAME FROM ACCOUNT

**CRITERIA**:

1. **List<?> list= session.createCriteria(Account.class)**
2. **.setProjection(Projections.property("name"))**
3. **.list();**

**(OR)**

1. **List<?> list= session.createCriteria(Account.class)**
2. **.setProjection(Property.forName("name"))**
3. **.list();**

**Example**:

**SQL**: SELECT COUNT(\*) FROM ACCOUNT

**CRITERIA**:

1. **List<?> list= session.createCriteria(Account.class)**
2. **.setProjection(Projections.rowCount())**
3. **.list();**
4. **sop(“Number of rows : “+list.get(0));**

(OR)

1. **Long rowCount = (Long) session.createCriteria(Account.class)**
2. **.setProjection(Projections.rowCount())**
3. **.uniqueResult();**
4. **Sop(“rowCount : “ + rowCount);**

**Example**:

**SQL**: SELECT COUNT(ACCNO) FROM ACCOUNT

**CRITERIA**:

1. **List<?> list= session.createCriteria(Account.class)**
2. **.setProjection(Projections.count("accountId"))**
3. **.list();**
4. **sop("count : "+list.get(0));**

**(OR)**

1. **Long count =(Long) session.createCriteria(Account.class)**
2. **.setProjection(Projections.count("accountId"))**
3. **.uniqueResult();**
4. **sop("count : "+count);**

**(OR)**

1. **Long count =(Long) session.createCriteria(Account.class)**
2. **.setProjection(Property.forName("accountId").count())**
3. **.uniqueResult();**
4. **sop("count : "+count);**

**Example**:

**SQL**: SELECT DISTINCT(NAME) FROM ACCOUNT

**CRITERIA**:

1. **List<?> list= session.createCriteria(Account.class)**
2. **.setProjection(**
3. **Projections.distinct(**
4. **Projections.property("name")))**
5. **.list();**
6. **sop("distinct Names : "+list);**

**Example**:

**SQL**: SELECT COUNT(DISTINCT NAME) FROM ACCOUNT

**CRITERIA**:

1. **List<?> list= session.createCriteria(Account.class)**
2. **.setProjection(Projections.countDistinct("name"))**
3. **.list();**
4. **sop("count of distinct Names : "+list.get(0));**

**(OR)**

1. **Long count =(Long) session.createCriteria(Account.class)**
2. **.setProjection(Projections.countDistinct("name"))**
3. **.uniqueResult();**
4. **sop("count of distinct Names : "+count);**

**Example**:

**SQL**: SELECT ACCNO FROM ACCOUNT

**CRITERIA**:

1. **List<?> list = session.createCriteria(Account.class)**
2. **.setProjection(Projections.id())**
3. **.list();**

**Example**:

**SQL**: SELECT MAX(BALANCE), AVG(BALANCE), MIN(BLAANCE), SUM(BALANCE) FROM ACCOUNT

**CRITERIA**:

1. **List<?> list = session.createCriteria(Account.class)**
2. **.setProjection(**
3. **Projections.projectionList()**
4. **.add(Projections.max("balance"))**
5. **.add(Projections.avg("balance"))**
6. **.add(Projections.min("balance"))**
7. **.add(Projections.sum("balance")))**
8. **.list();**

OR

1. **Property balanceProperty = Property.forName("balance");**
2. **List<?> list = session.createCriteria(Account.class)**
3. **.setProjection(**
4. **Projections.projectionList()**
5. **.add(balanceProperty.max())**
6. **.add(balanceProperty.avg())**
7. **.add(balanceProperty.min()))**
8. **.list();**

**Example**:

**SQL**: SELECT COUNT(ACCNO), NAME FROM ACCOUNT GROUP BY NAME

**CRITERIA**:

1. **List<?> list = session.createCriteria(Account.class)**
2. **.setProjection(**
3. **Projections.projectionList()**
4. **.add(Projections.count("accountId"))**
5. **.add(Projections.groupProperty("name")))**
6. **.list();**

OR

1. **List<?> list = session.createCriteria(Account.class)**
2. **.setProjection(**
3. **Projections.projectionList()**
4. **.add(Projections.count("accountId"))**
5. **.add(Property.forName("name").group()))**
6. **.list();**

**Example**:

**SQL**: SELECT ACCNO, NAME FROM ACCOUNT

WHERE

(BALANCE BETWEEN 2000 AND 4000)

AND

(

NAME LIKE ‘%yellareddy%’

OR

NAME LIKE ‘%sekhar%’

OR

NAME = ‘%kesavareddy%’

)

ORDER BY BALANCE DESC

**CRITERIA**:

1. **List<?> list = session.createCriteria(Account.class)**
2. **.setProjection(**
3. **Projections.projectionList()**
4. **.add(Projections.property("accountId"))**
5. **.add(Projections.property("name")))**
6. **.add(Restrictions.between("balance", 2000.0, 4000.0))**
7. **.add(Restrictions.disjunction()**
8. **.add(Restrictions.like("name", "yellareddy",MatchMode.ANYWHERE))**
9. **.add(Restrictions.like("name", "sekhar",MatchMode.ANYWHERE))**
10. **.add(Restrictions.like("name", "kesavareddy",MatchMode.ANYWHERE)))**
11. **.addOrder(Order.desc("balance"))**
12. **.list();**

**Example: Alias Names**

An alias can be assigned to a projection so that the projected value can be referred to in restrictions or orderings. Here are two different ways to do this:

1. **List results = session.createCriteria(Account.class)**
2. **.setProjection( Projections.alias( Projections.groupProperty("name"), "nm" ) )**
3. **.addOrder( Order.asc("nm") )**
4. **.list();**

**OR**

1. **List results = session.createCriteria(Account.class)**
2. **.setProjection( Projections.groupProperty("name").as("nm") )**
3. **.addOrder( Order.asc("nm") )**
4. **.list();**

The alias() and as() methods simply wrap a projection instance in another, aliased, instance of Projection. As a shortcut, you can assign an alias when you add the projection to a projection list:

1. **List results = session.createCriteria(Account.class)**
2. **.setProjection( Projections.projectionList()**
3. **.add( Projections.rowCount(), "accountCountByName" )**
4. **.add( Projections.avg("balance"), " avgBalance" )**
5. **.add( Projections.max("balance"), "maxBalance" )**
6. **.add( Projections.groupProperty("name"), "nm" )**
7. **)**
8. **.addOrder( Order.desc("accountCountByName") )**
9. **.addOrder( Order.desc("avgBalance") )**
10. **.list();**

**OR**

1. **List results = session.createCriteria(Account.class)**
2. **.setProjection( Projections.projectionList()**
3. **.add( Projections.rowCount().as("accountCountByName ") )**
4. **.add( Property.forName("balance ").avg().as("avgBalance ") )**
5. **.add( Property.forName("balance ").max().as("maxBalance ") )**
6. **.add( Property.forName("name").group().as("nm" )**
7. **)**
8. **.addOrder( Order.desc("accountCountByName ") )**
9. **.addOrder( Order.desc("avgBalance ") )**
10. **.list();**

**Example: Mapping criteria results to map**

1. **Map<String, Object> map = (Map<String, Object>)**
2. **session.createCriteria(Account.class)**
3. **.setProjection(**
4. **Projections.projectionList()**
5. **.add(Projections.avg("balance"), "avgBalance")**
6. **.add(Projections.min("balance"), "minBalance")**
7. **.add(Projections.max("balance"), "maxBalance")**
8. **.add(Projections.sum("balance"), "sumBalance"))**
9. **.setResultTransformer(Transformers.ALIAS\_TO\_ENTITY\_MAP)**
10. **.uniqueResult();**

**Example: Mapping criteria results to map**

1. **List<Map<String, Object>> list =**
2. **session.createCriteria(Account.class)**
3. **.setProjection(**
4. **Projections.projectionList()**
5. **.add(Projections.property("accountId"), "accountId")**
6. **.add(Projections.property("name"), "name")**
7. **.add(Projections.property("balance"), "balance"))**
8. **.setResultTransformer(Transformers.ALIAS\_TO\_ENTITY\_MAP)**
9. **.list();**

**Example: Mapping criteria results to list**

1. **List<List<Object>> list = session.createCriteria(Account.class)**
2. **.setProjection(**
3. **Projections.projectionList()**
4. **.add(Projections.property("accountId"), "accountId")**
5. **.add(Projections.property("name"), "name")**
6. **.add(Projections.property("balance"), "balance"))**
7. **.setResultTransformer(Transformers.TO\_LIST)**
8. **.list();**

**Example: Mapping criteria results to bean**

1. **List<Account> list = session.createCriteria(Account.class)**
2. **.setProjection(**
3. **Projections.projectionList()**
4. **.add(Projections.property("accountId"), "accountId")**
5. **.add(Projections.property("name"), "name")**
6. **.add(Projections.property("balance"), "balance"))**
7. **.setResultTransformer(Transformers.aliasToBean(Account.class))**
8. **.list();**

**DetachedCriteria**

A **DetachedCriteria** can also be used to express a subquery. **Criterion** instances involving subqueries can be obtained via **Subqueries** or **Property**.

**Example**:

**SQL**: SELECT \* FROM ACCOUNT WHERE BALANCE> ( SELECT AVG(BALANCE) FROM ACCOUNT)

**CRITERIA**:

1. **DetachedCriteria dCriteria = DetachedCriteria.forClass(Account.class);**
2. **Projection avgProjection = Projections.avg("balance");**
3. **ProjectionList pList = Projections.projectionList();**
4. **pList.add(avgProjection);**
5. **dCriteria.setProjection(pList);**
7. **Criteria criteria = session.createCriteria(Account.class);**
8. **Property payProperty= Property.forName(“balance");**
9. **Criterion criterion = payProperty.gt(dCriteria);**
11. **criteria.add(criterion);**
12. **List<Account> list = criteria.list();**

**Example**:

**SQL**: SELECT \* FROM ACCOUNT WHERE > ( SELECT AVG(BALANCE) FROM ACCOUNT)

**CRITERIA**:

1. **DetachedCriteria requestedNames =**
2. **DetachedCriteria.forClass(Account.class)**
3. **.setProjection(Projections.property("name"))**
4. **.add(Restrictions.gt("accountId", 5005));**
6. **Criteria criteria = session.createCriteria(Account.class);**
7. **criteria.add(Subqueries.in("name", requestedNames));**
8. **//criteria.add(Property.forName("name").in(requestedNames));**
9. **List<Account> companies = criteria.list();**

**Conclusion**

* **HQL** is suitable for executing **Static Queries**, where as **Criteria** is suitable for executing **Dynamic Queries**
* **Criteria** used to take more time to execute than **HQL**
* Nothing is perfect, do consider your **project needs** and use it wisely.