1. **What is ORM ?**

ORM stands for object/relational mapping. ORM is the automated persistence of objects in a Java application to the tables in a relational database.

|  |
| --- |
|  |

1. **What does ORM consists of ?**

An ORM solution consists of the followig four pieces:

* API for performing basic CRUD operations
* API to express queries refering to classes
* Facilities to specify metadata
* Optimization facilities : dirty checking,lazy associations fetching

1. **What are the ORM levels ?**

The ORM levels are:

* Pure relational (stored procedure.)
* Light objects mapping (JDBC)
* Medium object mapping
* Full object Mapping (composition,inheritance, polymorphism, persistence by reachability)

1. **What is Hibernate?**

Hibernate is a pure Java object-relational mapping (ORM) and persistence framework that allows you to map plain old Java objects to relational database tables using (XML) configuration files.Its purpose is to relieve the developer from a significant amount of relational data persistence-related programming tasks.

1. **Why do you need ORM tools like hibernate?**

The main advantage of ORM like hibernate is that it shields developers from messy SQL. Apart from this, ORM provides following benefits:

* **Improved productivity**
  + High-level object-oriented API
  + Less Java code to write
  + No SQL to write
* **Improved performance**
  + Sophisticated caching
  + Lazy loading
  + Eager loading
* **Improved maintainability**
  + A lot less code to write
* **Improved portability**
  + ORM framework generates database-specific SQL for you

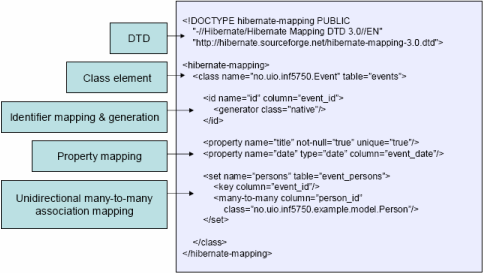
1. **What Does Hibernate Simplify?**

Hibernate simplifies:

* Saving and retrieving your domain objects
* Making database column and table name changes
* Centralizing pre save and post retrieve logic
* Complex joins for retrieving related items
* Schema creation from object model

1. **What is the need for Hibernate xml mapping file?**

Hibernate mapping file tells Hibernate which tables and columns to use to load and store objects. Typical mapping file look as follows:



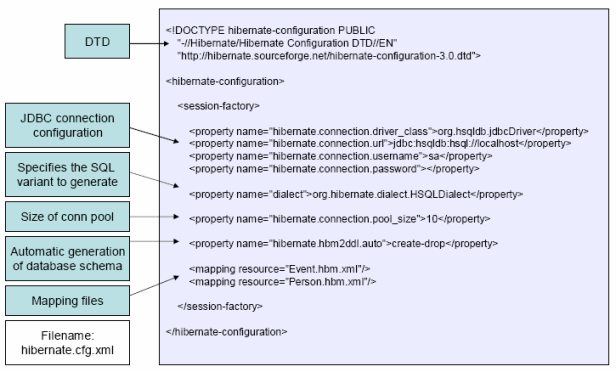
**8. What are the most common methods of Hibernate configuration?**

The most common methods of Hibernate configuration are:

* Programmatic configuration
* XML configuration (hibernate.cfg.xml)

**9. What are the important tags of hibernate.cfg.xml?**

Following are the important tags of hibernate.cfg.xml:

  
  
**10. What are the Core interfaces are of Hibernate framework?**

The five core interfaces are used in just about every Hibernate application. Using these interfaces, you can store and retrieve persistent objects and control transactions.

* Session interface
* SessionFactory interface
* Configuration interface
* Transaction interface
* Query and Criteria interfaces

**11. What role does the Session interface play in Hibernate?**

The Session interface is the primary interface used by Hibernate applications. It is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It allows you to create query objects to retrieve persistent objects.  
  
Session session = sessionFactory.openSession();

**Session interface role**:

* Wraps a JDBC connection
* Factory for Transaction
* Holds a mandatory (first-level) cache of persistent objects, used when navigating the object graph or looking up objects by identifier

**12. What role does the SessionFactory interface play in Hibernate?**

The application obtains Session instances from a SessionFactory. There is typically a single SessionFactory for the whole applicationå¹¼reated during application initialization. The SessionFactory caches generate SQL statements and other mapping metadata that Hibernate uses at runtime. It also holds cached data that has been read in one unit of work and may be reused in a future unit of work  
  
SessionFactory sessionFactory = configuration.buildSessionFactory();

**13. What is the general flow of Hibernate communication with RDBMS?**

The general flow of Hibernate communication with RDBMS is :

* Load the Hibernate configuration file and create configuration object. It will automatically load all hbm mapping files
* Create session factory from configuration object
* Get one session from this session factory
* Create HQL Query
* Execute query to get list containing Java objects

**14. What is Hibernate Query Language (HQL)?**

Hibernate offers a query language that embodies a very powerful and flexible mechanism to query, store, update, and retrieve objects from a database. This language, the Hibernate query Language (HQL), is an object-oriented extension to SQL.

**15. How do you map Java Objects with Database tables?**

* First we need to write Java domain objects (beans with setter and getter).
* Write hbm.xml, where we map java class to table and database columns to Java class variables.

**Example** :

<hibernate-mapping>  
  <class name="com.test.User"  table="user">  
   <property  column="USER\_NAME" length="255"   
      name="userName" not-null="true"  type="java.lang.String"/>  
   <property  column="USER\_PASSWORD" length="255"  
 name="userPassword" not-null="true"  type="java.lang.String"/>  
 </class>  
</hibernate-mapping>

**16.Whatï¿½s the difference between load() and get()?**

load() vs. get() :-

|  |  |
| --- | --- |
| **load()** | **get()** |
| Only use the load() method if you are sure that the object exists. | If you are not sure that the object exists, then use one of the get() methods. |
| load() method will throw an exception if the unique id is not found in the database. | get() method will return null if the unique id is not found in the database. |
| load() just returns a proxy by default and database wonï¿½t be hit until the proxy is first invoked. | get() will hit the database immediately. |

**17. What is the difference between and merge and update ?**

update(Object object)

Used to **reattach a detached object** (i.e., an object that was once persistent but the session was closed) back into a new Hibernate session.

 Requires that **no other instance with the same identifier** is already associated with the session (otherwise it throws NonUniqueObjectException).

 Directly updates the database with the values of the object.

 **Does not return a new object.**

 Fast and simple if you’re sure the object is not already attached.

merge(Object object)

Used to **merge the state** of a detached object into the current persistence context. It **returns a new managed instance** (the one that is actually managed by the session).

 Safe to use when there may already be an instance with the same ID in the session.

 Copies values from the detached object to a managed instance.

 **Returns a managed instance** (not necessarily the same object you passed).

 Does not throw NonUniqueObjectException.

**18.How do you define sequence generated primary key in hibernate?**

Using <generator> tag.  
**Example**:-

<id column="USER\_ID" name="id" type="java.lang.Long">   
 <generator class="sequence">   
 <param name="table">SEQUENCE\_NAME</param>  
 <generator>  
</id>

**19. Define cascade and inverse option in one-many mapping?**

cascade - enable operations to cascade to child entities.  
cascade="all|none|save-update|delete|all-delete-orphan"  
  
inverse - mark this collection as the "inverse" end of a bidirectional association.  
inverse="true|false"   
Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

**20. What do you mean by Named ï¿½ SQL query?**

Named SQL queries are defined in the mapping xml document and called wherever required.  
**Example:**

<sql-query name = "empdetails">  
   <return alias="emp" class="com.test.Employee"/>  
      SELECT emp.EMP\_ID AS {emp.empid},  
                 emp.EMP\_ADDRESS AS {emp.address},  
                 emp.EMP\_NAME AS {emp.name}   
 FROM Employee EMP WHERE emp.NAME LIKE :name  
</sql-query>

Invoke Named Query :

List people = session.getNamedQuery("empdetails")  
 .setString("TomBrady", name)  
 .setMaxResults(50)  
 .list();

**21. How do you invoke Stored Procedures?**

<sql-query name="selectAllEmployees\_SP" callable="true">  
 <return alias="emp" class="employee">  
  <return-property name="empid" column="EMP\_ID"/>

<return-property name="name" column="EMP\_NAME"/>         
 <return-property name="address" column="EMP\_ADDRESS"/>  
    { ? = call selectAllEmployees() }  
 </return>  
</sql-query>

**22. Explain Criteria API**

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.  
**Example** :

List employees = session.createCriteria(Employee.class)  
        .add(Restrictions.like("name", "a%") )  
          .add(Restrictions.like("address", "Boston"))  
 .addOrder(Order.asc("name") )  
 .list();

**23. Define HibernateTemplate?**

org.springframework.orm.hibernate.HibernateTemplate is a helper class which provides different methods for querying/retrieving data from the database. It also converts checked HibernateExceptions into unchecked DataAccessExceptions.

**24. What are the benefits does HibernateTemplate provide?**

The benefits of HibernateTemplate are :

* HibernateTemplate, a Spring Template class simplifies interactions with Hibernate Session.
* Common functions are simplified to single method calls.
* Sessions are automatically closed.
* Exceptions are automatically caught and converted to runtime exceptions.

**25. How do you switch between relational databases without code changes?**

Using Hibernate SQL Dialects , we can switch databases. Hibernate will generate appropriate hql queries based on the dialect defined.

**26. If you want to see the Hibernate generated SQL statements on console, what should we do?**

In Hibernate configuration file set as follows:   
<property name="show\_sql">true</property>

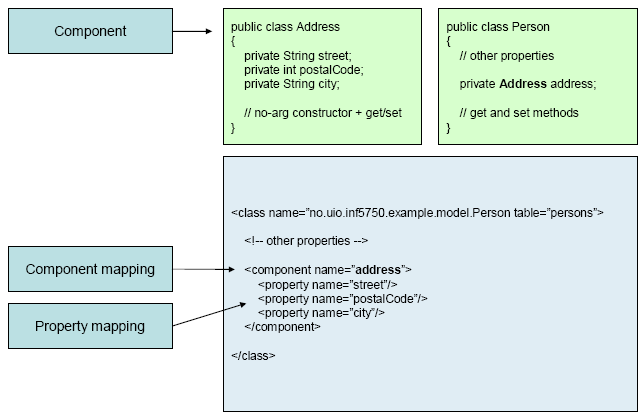
**27. What are derived properties?**

The properties that are not mapped to a column, but calculated at runtime by evaluation of an expression are called derived properties. The expression can be defined using the formula attribute of the element.

**28. What is component mapping in Hibernate?**

* A component is an object saved as a value, not as a reference
* A component can be saved directly without needing to declare interfaces or identifier properties
* Required to define an empty constructor
* Shared references not supported

**Example**:



**29. What is the difference between sorted and ordered collection in hibernate?**

**sorted collection vs. order collection** :-

|  |  |
| --- | --- |
| **sorted collection** | **order collection** |
| A sorted collection is sorting a collection by utilizing the sorting features provided by the Java collections framework. The sorting occurs in the memory of JVM which running Hibernate, after the data being read from database using java comparator. | Order collection is sorting a collection by specifying the order-by clause for sorting this collection when retrieval. |
| If your collection is not large, it will be more efficient way to sort it. | If your collection is very large, it will be more efficient way to sort it . |

**31. What is the advantage of Hibernate over jdbc?**

Hibernate Vs. JDBC :-

|  |  |
| --- | --- |
| **JDBC** | **Hibernate** |
| With JDBC, developer has to write code to map an object model's data representation to a relational data model and its corresponding database schema. | Hibernate is flexible and powerful ORM solution to map Java classes to database tables. Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this. |
| With JDBC, the automatic mapping of Java objects with database tables and vice versa conversion is to be taken care of by the developer manually with lines of code. | Hibernate provides transparent persistence and developer does not need to write code explicitly to map database tables tuples to application objects during interaction with RDBMS. |
| JDBC supports only native Structured Query Language (SQL). Developer has to find out the efficient way to access database, i.e. to select effective query from a number of queries to perform same task. | Hibernate provides a powerful query language Hibernate Query Language (independent from type of database) that is expressed in a familiar SQL like syntax and includes full support for polymorphic queries. Hibernate also supports native SQL statements. It also selects an effective way to perform a database manipulation task for an application. |
| Application using JDBC to handle persistent data (database tables) having database specific code in large amount. The code written to map table data to application objects and vice versa is actually to map table fields to object properties. As table changed or database changed then it’s essential to change object structure as well as to change code written to map table-to-object/object-to-table. | Hibernate provides this mapping itself. The actual mapping between tables and application objects is done in XML files. If there is change in Database or in any table then the only need to change XML file properties. |
| With JDBC, it is developer’s responsibility to handle JDBC result set and convert it to Java objects through code to use this persistent data in application. So with JDBC, mapping between Java objects and database tables is done manually. | Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects. It relieves programmer from manual handling of persistent data, hence reducing the development time and maintenance cost. |
| With JDBC, caching is maintained by hand-coding. | Hibernate, with Transparent Persistence, cache is set to application work space. Relational tuples are moved to this cache as a result of query. It improves performance if client application reads same data many times for same write. Automatic Transparent Persistence allows the developer to concentrate more on business logic rather than this application code. |
| In JDBC there is no check that always every user has updated data. This check has to be added by the developer. | Hibernate enables developer to define version type field to application, due to this defined field Hibernate updates version field of database table every time relational tuple is updated in form of Java class object to that table. So if two users retrieve same tuple and then modify it and one user save this modified tuple to database, version is automatically updated for this tuple by Hibernate. When other user tries to save updated tuple to database then it does not allow saving it because this user does not have updated data. |

**32. What are the Collection types in Hibernate ?**

* Bag
* Set
* List
* Array
* Map

**33. What are the ways to express joins in HQL?**

HQL provides four ways of expressing (inner and outer) joins:-

* An *implicit* association join
* An ordinary join in the FROM clause
* A fetch join in the FROM clause.
* A *theta-style* join in the WHERE clause.

**34. Define cascade and inverse option in one-many mapping?**

cascade - enable operations to cascade to child entities.  
cascade="all|none|save-update|delete|all-delete-orphan"  
  
inverse - mark this collection as the "inverse" end of a bidirectional association.  
inverse="true|false"   
Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

**35. What is Hibernate proxy?**

The proxy attribute enables lazy initialization of persistent instances of the class. Hibernate will initially return CGLIB proxies which implement the named interface. The actual persistent object will be loaded when a method of the proxy is invoked.

**36. How can Hibernate be configured to access an instance variable directly and not through a setter method ?**

By mapping the property with access="field" in Hibernate metadata. This forces hibernate to bypass the setter method and access the instance variable directly while initializing a newly loaded object.

**37. How can a whole class be mapped as immutable?**

Mark the class as mutable="false" (Default is true),. This specifies that instances of the class are (not) mutable. Immutable classes, may not be updated or deleted by the application.

**38. What is the use of dynamic-insert and dynamic-update attributes in a class mapping?**

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.

* dynamic-update (defaults to false): Specifies that UPDATE SQL should be generated at runtime and contain only those columns whose values have changed
* dynamic-insert (defaults to false): Specifies that INSERT SQL should be generated at runtime and contain only the columns whose values are not null.

**39. What do you mean by fetching strategy ?**

A *fetching strategy* is the strategy Hibernate will use for retrieving associated objects if the application needs to navigate the association. Fetch strategies may be declared in the O/R mapping metadata, or over-ridden by a particular HQL or Criteria query.

**40. What is automatic dirty checking?**

Automatic dirty checking is a feature that saves us the effort of explicitly asking Hibernate to update the database when we modify the state of an object inside a transaction.

**41. What is transactional write-behind?**

Hibernate uses a sophisticated algorithm to determine an efficient ordering that avoids database foreign key constraint violations but is still sufficiently predictable to the user. This feature is called transactional write-behind.

**42. What are Callback interfaces?**

Callback interfaces allow the application to receive a notification when something interesting happens to an object—for example, when an object is loaded, saved, or deleted. Hibernate applications don't need to implement these callbacks, but they're useful for implementing certain kinds of generic functionality.  
**43. What are the types of Hibernate instance states ?**

Three types of instance states:

* Transient -The instance is not associated with any persistence context
* Persistent -The instance is associated with a persistence context
* Detached -The instance was associated with a persistence context which has been closed – currently not associated  
  **44.What are the differences between EJB 3.0 & Hibernate**

|  |  |
| --- | --- |
| **Hibernate** | **EJB 3.0** |
| **Session**–Cache or collection of loaded objects relating to a single unit of work | **Persistence Context**-Set of entities that can be managed by a given EntityManager is defined by a persistence unit |
| **XDoclet Annotations** used to support Attribute Oriented Programming | **Java 5.0 Annotations** used to support Attribute Oriented Programming |
| **Defines HQL** for expressing queries to the database | **Defines EJB QL** for expressing queries |
| **Supports Entity Relationships** through mapping files and annotations in JavaDoc | **Support Entity Relationships** through Java 5.0 annotations |
| **Provides a Persistence Manager API** exposed via the Session, Query, Criteria, and Transaction API | **Provides and Entity Manager Interface** for managing CRUD operations for an Entity |
| **Provides callback support** through lifecycle, interceptor, and validatable interfaces | **Provides callback support** through Entity Listener and Callback methods |
| **Entity Relationships are unidirectional**. Bidirectional relationships are implemented by two unidirectional relationships | **Entity Relationships are bidirectional or unidirectional** |

Hibernate Vs EJB 3.0 :-

**45.What are the types of inheritance models in Hibernate?**

There are three types of inheritance models in Hibernate:

* Table per class hierarchy
* Table per subclass
* Table per concrete class

======================================================================================

### What is Hibernate Framework?

**Object-relational mapping** or ORM is the programming technique to map application domain model objects to the relational database tables. Hibernate is java based ORM tool that provides framework for mapping application domain objects to the relational database tables and vice versa.

Hibernate provides reference implementation of Java Persistence API, that makes it a great choice as ORM tool with benefits of loose coupling. We can use Hibernate persistence API for CRUD operations. Hibernate framework provide option to map plain old java objects to traditional database tables with the use of JPA annotations as well as XML based configuration.

Similarly hibernate configurations are flexible and can be done from XML configuration file as well as programmatically. For a quick overview of hibernate framework usage, you can go through [Hibernate Beginners Tutorial](http://www.journaldev.com/2882/hibernate-tutorial-for-beginners-using-xml-annotations-and-property-configurations).

### What is Java Persistence API (JPA)?

Java Persistence API (JPA) provides specification for managing the relational data in applications. Current JPA version 2.1 was started in July 2011 as JSR 338. JPA 2.1 was approved as final on 22 May 2013.

JPA specifications is defined with annotations in javax.persistence package. Using JPA annotation helps us in writing implementation independent code.

### What are the important benefits of using Hibernate Framework?

Some of the important benefits of using hibernate framework are:

* 1. Hibernate eliminates all the boiler-plate code that comes with JDBC and takes care of managing resources, so we can focus on business logic.
  2. Hibernate framework provides support for XML as well as JPA annotations, that makes our code implementation independent.
  3. Hibernate provides a powerful query language (HQL) that is similar to SQL. However, HQL is fully object-oriented and understands concepts like inheritance, polymorphism and association.
  4. Hibernate is an open source project from Red Hat Community and used worldwide. This makes it a better choice than others because learning curve is small and there are tons of online documentations and help is easily available in forums.
  5. Hibernate is easy to integrate with other Java EE frameworks, it’s so popular that Spring Framework provides built-in support for integrating hibernate with Spring applications.
  6. Hibernate supports lazy initialization using proxy objects and perform actual database queries only when it’s required.
  7. Hibernate cache helps us in getting better performance.
  8. For database vendor specific feature, hibernate is suitable because we can also execute native sql queries.

Overall hibernate is the best choice in current market for ORM tool, it contains all the features that you will ever need in an ORM tool.

### What are the advantages of Hibernate over JDBC?

Some of the important advantages of Hibernate framework over JDBC are:

* 1. Hibernate removes a lot of boiler-plate code that comes with JDBC API, the code looks more cleaner and readable.
  2. Hibernate supports inheritance, associations and collections. These features are not present with JDBC API.
  3. Hibernate implicitly provides transaction management, in fact most of the queries can’t be executed outside transaction. In JDBC API, we need to write code for transaction management using commit and rollback. Read more at [JDBC Transaction Management](http://www.journaldev.com/2483/jdbc-transaction-management-and-savepoint-example-tutorial).
  4. JDBC API throws SQLException that is a checked exception, so we need to write a lot of try-catch block code. Most of the times it’s redundant in every JDBC call and used for transaction management. Hibernate wraps JDBC exceptions and throw JDBCException or HibernateException un-checked exception, so we don’t need to write code to handle it. Hibernate built-in transaction management removes the usage of try-catch blocks.
  5. Hibernate Query Language (HQL) is more object oriented and close to java programming language. For JDBC, we need to write native sql queries.
  6. Hibernate supports caching that is better for performance, JDBC queries are not cached hence performance is low.
  7. Hibernate provide option through which we can create database tables too, for JDBC tables must exist in the database.
  8. Hibernate configuration helps us in using JDBC like connection as well as JNDI DataSource for connection pool. This is very important feature in enterprise application and completely missing in JDBC API.
  9. Hibernate supports JPA annotations, so code is independent of implementation and easily replaceable with other ORM tools. JDBC code is very tightly coupled with the application.

### Name some important interfaces of Hibernate framework?

Some of the important interfaces of Hibernate framework are:

* 1. **SessionFactory (org.hibernate.SessionFactory)**: SessionFactory is an immutable thread-safe cache of compiled mappings for a single database. We need to initialize SessionFactory once and then we can cache and reuse it. SessionFactory instance is used to get the Session objects for database operations.
  2. **Session (org.hibernate.Session)**: Session is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It wraps JDBC java.sql.Connectionand works as a factory for org.hibernate.Transaction. We should open session only when it’s required and close it as soon as we are done using it. Session object is the interface between java application code and hibernate framework and provide methods for CRUD operations.
  3. **Transaction (org.hibernate.Transaction)**: Transaction is a single-threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC or JTA transaction. A org.hibernate.Session might span multiple org.hibernate.Transaction in some cases.

### What is hibernate configuration file?

Hibernate configuration file contains database specific configurations and used to initialize SessionFactory. We provide database credentials or JNDI resource information in the hibernate configuration xml file. Some other important parts of hibernate configuration file is Dialect information, so that hibernate knows the database type and mapping file or class details.

### What is hibernate mapping file?

Hibernate mapping file is used to define the entity bean fields and database table column mappings. We know that JPA annotations can be used for mapping but sometimes XML mapping file comes handy when we are using third party classes and we can’t use annotations.

### Name some important annotations used for Hibernate mapping?

Hibernate supports JPA annotations and it has some other annotations in org.hibernate.annotationspackage. Some of the important JPA and hibernate annotations used are:

* 1. **javax.persistence.Entity**: Used with model classes to specify that they are entity beans.
  2. **javax.persistence.Table**: Used with entity beans to define the corresponding table name in database.
  3. **javax.persistence.Access**: Used to define the access type, either field or property. Default value is field and if you want hibernate to use getter/setter methods then you need to set it to property.
  4. **javax.persistence.Id**: Used to define the primary key in the entity bean.
  5. **javax.persistence.EmbeddedId**: Used to define composite primary key in the entity bean.
  6. **javax.persistence.Column**: Used to define the column name in database table.
  7. **javax.persistence.GeneratedValue**: Used to define the strategy to be used for generation of primary key. Used in conjunction with javax.persistence.GenerationType enum.
  8. **javax.persistence.OneToOne**: Used to define the one-to-one mapping between two entity beans. We have other similar annotations as OneToMany, ManyToOne and ManyToMany
  9. **org.hibernate.annotations.Cascade**: Used to define the cascading between two entity beans, used with mappings. It works in conjunction with org.hibernate.annotations.CascadeType
  10. **javax.persistence.PrimaryKeyJoinColumn**: Used to define the property for foreign key. Used withorg.hibernate.annotations.GenericGenerator and org.hibernate.annotations.Parameter

Here are two classes showing usage of these annotations.

|  |  |
| --- | --- |
| Employee.java | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | package com.journaldev.hibernate.model;    import javax.persistence.Access;  import javax.persistence.AccessType;  import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.GenerationType;  import javax.persistence.Id;  import javax.persistence.OneToOne;  import javax.persistence.Table;    import org.hibernate.annotations.Cascade;    @Entity  @Table(name = "EMPLOYEE")  @Access(value=AccessType.FIELD)  public class Employee {        @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      @Column(name = "emp\_id")      private long id;        @Column(name = "emp\_name")      private String name;        @OneToOne(mappedBy = "employee")      @Cascade(value = org.hibernate.annotations.CascadeType.ALL)      private Address address;        //getter setter methods  } |
| Address.java | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | package com.journaldev.hibernate.model;    import javax.persistence.Access;  import javax.persistence.AccessType;  import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.Id;  import javax.persistence.OneToOne;  import javax.persistence.PrimaryKeyJoinColumn;  import javax.persistence.Table;    import org.hibernate.annotations.GenericGenerator;  import org.hibernate.annotations.Parameter;    @Entity  @Table(name = "ADDRESS")  @Access(value=AccessType.FIELD)  public class Address {        @Id      @Column(name = "emp\_id", unique = true, nullable = false)      @GeneratedValue(generator = "gen")      @GenericGenerator(name = "gen", strategy = "foreign", parameters = { @Parameter(name = "property", value = "employee") })      private long id;        @Column(name = "address\_line1")      private String addressLine1;        @OneToOne      @PrimaryKeyJoinColumn      private Employee employee;        //getter setter methods  } | |

### What is Hibernate SessionFactory and how to configure it?

SessionFactory is the factory class used to get the Session objects. SessionFactory is responsible to read the hibernate configuration parameters and connect to the database and provide Session objects. Usually an application has a single SessionFactory instance and threads servicing client requests obtain Session instances from this factory.

The internal state of a SessionFactory is immutable. Once it is created this internal state is set. This internal state includes all of the metadata about Object/Relational Mapping.

SessionFactory also provide methods to get the Class metadata and Statistics instance to get the stats of query executions, second level cache details etc.

### Hibernate SessionFactory is thread safe?

Internal state of SessionFactory is immutable, so it’s thread safe. Multiple threads can access it simultaneously to get Session instances.

### What is Hibernate Session and how to get it?

Hibernate Session is the interface between java application layer and hibernate. This is the core interface used to perform database operations. Lifecycle of a session is bound by the beginning and end of a transaction.

Session provide methods to perform create, read, update and delete operations for a persistent object. We can execute HQL queries, SQL native queries and create criteria using Session object.

### Hibernate Session is thread safe?

Hibernate Session object is not thread safe, every thread should get it’s own session instance and close it after it’s work is finished.

### What is difference between openSession and getCurrentSession?

Hibernate SessionFactory getCurrentSession() method returns the session bound to the context. But for this to work, we need to configure it in hibernate configuration file. Since this session object belongs to the hibernate context, we don’t need to close it. Once the session factory is closed, this session object gets closed.s

|  |  |
| --- | --- |
| 1 | <property name="hibernate.current\_session\_context\_class">thread</property> |

Hibernate SessionFactory openSession() method always opens a new session. We should close this session object once we are done with all the database operations. We should open a new session for each request in multi-threaded environment.

There is another method openStatelessSession() that returns stateless session, for more details with examples please read [Hibernate openSession vs getCurrentSession](http://www.journaldev.com/3522/hibernate-sessionfactory-opensession-vs-getcurrentsession-vs-openstatelesssession).

### What is difference between Hibernate Session get() and load() method?

Hibernate session comes with different methods to load data from database. get and load are most used methods, at first look they seems similar but there are some differences between them.

* 1. get() loads the data as soon as it’s called whereas load() returns a proxy object and loads data only when it’s actually required, so load() is better because it support lazy loading.
  2. Since load() throws exception when data is not found, we should use it only when we know data exists.
  3. We should use get() when we want to make sure data exists in the database.

For clarification regarding the differences, please read [Hibernate get vs load](http://www.journaldev.com/3472/hibernate-session-get-vs-load-difference-with-examples).

### What is hibernate caching? Explain Hibernate first level cache?

As the name suggests, hibernate caches query data to make our application faster. Hibernate Cache can be very useful in gaining fast application performance if used correctly. The idea behind cache is to reduce the number of database queries, hence reducing the throughput time of the application.

Hibernate first level cache is associated with the Session object. Hibernate first level cache is enabled by default and there is no way to disable it. However hibernate provides methods through which we can delete selected objects from the cache or clear the cache completely.  
Any object cached in a session will not be visible to other sessions and when the session is closed, all the cached objects will also be lost.

For better explanation, please read [Hibernate First Level Cache](http://www.journaldev.com/2969/hibernate-first-level-cache-example-with-explanation).

### How to configure Hibernate Second Level Cache using EHCache?

EHCache is the best choice for utilizing hibernate second level cache. Following steps are required to enable EHCache in hibernate application.

* 1. Add hibernate-ehcache dependency in your maven project, if it’s not maven then add corresponding jars.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <dependency>          <groupId>org.hibernate</groupId>          <artifactId>hibernate-ehcache</artifactId>          <version>4.3.5.Final</version>  </dependency> |

* 1. Add below properties in hibernate configuration file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>    <!-- For singleton factory -->  <!-- <property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.SingletonEhCacheRegionFactory</property>  -->    <!-- enable second level cache and query cache -->  <property name="hibernate.cache.use\_second\_level\_cache">true</property>  <property name="hibernate.cache.use\_query\_cache">true</property>  <property name="net.sf.ehcache.configurationResourceName">/myehcache.xml</property> |

* 1. Create EHCache configuration file, a sample file would look like below.

|  |  |
| --- | --- |
| myehcache.xml | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | <?xml version="1.0" encoding="UTF-8"?>  <ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"      xsi:noNamespaceSchemaLocation="ehcache.xsd" updateCheck="true"      monitoring="autodetect" dynamicConfig="true">        <diskStore path="java.io.tmpdir/ehcache" />        <defaultCache maxEntriesLocalHeap="10000" eternal="false"          timeToIdleSeconds="120" timeToLiveSeconds="120" diskSpoolBufferSizeMB="30"          maxEntriesLocalDisk="10000000" diskExpiryThreadIntervalSeconds="120"          memoryStoreEvictionPolicy="LRU" statistics="true">          <persistence strategy="localTempSwap" />      </defaultCache>        <cache name="employee" maxEntriesLocalHeap="10000" eternal="false"          timeToIdleSeconds="5" timeToLiveSeconds="10">          <persistence strategy="localTempSwap" />      </cache>        <cache name="org.hibernate.cache.internal.StandardQueryCache"          maxEntriesLocalHeap="5" eternal="false" timeToLiveSeconds="120">          <persistence strategy="localTempSwap" />      </cache>        <cache name="org.hibernate.cache.spi.UpdateTimestampsCache"          maxEntriesLocalHeap="5000" eternal="true">          <persistence strategy="localTempSwap" />      </cache>  </ehcache> |

* 1. Annotate entity beans with @Cache annotation and caching strategy to use. For example,

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import org.hibernate.annotations.Cache;  import org.hibernate.annotations.CacheConcurrencyStrategy;    @Entity  @Table(name = "ADDRESS")  @Cache(usage=CacheConcurrencyStrategy.READ\_ONLY, region="employee")  public class Address {    } |

That’s it, we are done. Hibernate will use the EHCache for second level caching, read [Hibernate EHCache Example](http://www.journaldev.com/2980/hibernate-ehcache-second-level-caching-example-tutorial) for a complete example with explanation.

### What are different states of an entity bean?

An entity bean instance can exist is one of the three states.

* 1. **Transient**: When an object is never persisted or associated with any session, it’s in transient state. Transient instances may be made persistent by calling save(), persist() or saveOrUpdate(). Persistent instances may be made transient by calling delete().
  2. **Persistent**: When an object is associated with a unique session, it’s in persistent state. Any instance returned by a get() or load() method is persistent.
  3. **Detached**: When an object is previously persistent but not associated with any session, it’s in detached state. Detached instances may be made persistent by calling update(), saveOrUpdate(), lock() or replicate(). The state of a transient or detached instance may also be made persistent as a new persistent instance by calling merge().

### What is use of Hibernate Session merge() call?

Hibernate merge can be used to update existing values, however this method create a copy from the passed entity object and return it. The returned object is part of persistent context and tracked for any changes, passed object is not tracked. For example program, read [Hibernate merge](http://www.journaldev.com/3481/hibernate-save-vs-saveorupdate-vs-persist-vs-merge-vs-update-explanation-with-examples).

### What is difference between Hibernate save(), saveOrUpdate() and persist() methods?

Hibernate save can be used to save entity to database. Problem with save() is that it can be invoked without a transaction and if we have mapping entities, then only the primary object gets saved causing data inconsistencies. Also save returns the generated id immediately.

Hibernate persist is similar to save with transaction. I feel it’s better than save because we can’t use it outside the boundary of transaction, so all the object mappings are preserved. Also persist doesn’t return the generated id immediately, so data persistence happens when needed.

Hibernate saveOrUpdate results into insert or update queries based on the provided data. If the data is present in the database, update query is executed. We can use saveOrUpdate() without transaction also, but again you will face the issues with mapped objects not getting saved if session is not flushed. For example usage of these methods, read [Hibernate save vs persist](http://www.journaldev.com/3481/hibernate-save-vs-saveorupdate-vs-persist-vs-merge-vs-update-explanation-with-examples).

### What will happen if we don’t have no-args constructor in Entity bean?

Hibernate uses [Reflection API](http://www.journaldev.com/1789/java-reflection-tutorial-for-classes-methods-fields-constructors-annotations-and-much-more) to create instance of Entity beans, usually when you call get() or load() methods. The method Class.newInstance() is used for this and it requires no-args constructor. So if you won’t have no-args constructor in entity beans, hibernate will fail to instantiate it and you will getHibernateException.

### What is difference between sorted collection and ordered collection, which one is better?

When we use Collection API sorting algorithms to sort a collection, it’s called sorted list. For small collections, it’s not much of an overhead but for larger collections it can lead to slow performance and OutOfMemory errors. Also the entity beans should implement Comparable or Comparator interface for it to work, read more at [java object list sorting](http://www.journaldev.com/780/java-comparable-and-comparator-example-to-sort-objects).

If we are using Hibernate framework to load collection data from database, we can use it’s Criteria API to use “order by” clause to get ordered list. Below code snippet shows you how to get it.

|  |  |
| --- | --- |
| 1  2 | List<Employee> empList = session.createCriteria(Employee.class)                          .addOrder(Order.desc("id")).list(); |

Ordered list is better than sorted list because the actual sorting is done at database level, that is fast and doesn’t cause memory issues.

### What are the collection types in Hibernate?

There are five collection types in hibernate used for one-to-many relationship mappings.

* 1. Bag
  2. Set
  3. List
  4. Array
  5. Map

### How to implement Joins in Hibernate?

There are various ways to implement joins in hibernate.

* 1. Using associations such as one-to-one, one-to-many etc.
  2. Using JOIN in the HQL query. There is another form “join fetch” to load associated data simultaneously, no lazy loading.
  3. We can fire native sql query and use join keyword.

### Why we should not make Entity Class final?

Hibernate use proxy classes for lazy loading of data, only when it’s needed. This is done by extending the entity bean, if the entity bean will be final then lazy loading will not be possible, hence low performance.

### What is HQL and what are it’s benefits?

Hibernate Framework comes with a powerful object-oriented query language – Hibernate Query Language (HQL). It’s very similar to SQL except that we use Objects instead of table names, that makes it more close to object oriented programming.

Hibernate query language is case-insensitive except for java class and variable names. So SeLeCT is the same as sELEct is the same as SELECT, but com.journaldev.model.Employee is not same as com.journaldev.model.EMPLOYEE.

The HQL queries are cached but we should avoid it as much as possible, otherwise we will have to take care of associations. However it’s a better choice than native sql query because of Object-Oriented approach. Read more at [HQL Example](http://www.journaldev.com/2954/hibernate-query-language-hql-example-tutorial).

### What is Query Cache in Hibernate?

Hibernate implements a cache region for queries resultset that integrates closely with the hibernate second-level cache.

This is an optional feature and requires additional steps in code. This is only useful for queries that are run frequently with the same parameters. First of all we need to configure below property in hibernate configuration file.

|  |  |
| --- | --- |
| 1 | <property name="hibernate.cache.use\_query\_cache">true</property> |

And in code, we need to use setCacheable(true) method of Query, quick example looks like below.

|  |  |
| --- | --- |
| 1  2  3 | Query query = session.createQuery("from Employee");  query.setCacheable(true);  query.setCacheRegion("ALL\_EMP"); |

### Can we execute native sql query in hibernate?

Hibernate provide option to execute native SQL queries through the use of SQLQuery object.

For normal scenarios, it is however not the recommended approach because we loose benefits related to hibernate association and hibernate first level caching. Read more at [Hibernate Native SQL Query Example](http://www.journaldev.com/3422/hibernate-native-sql-example-addscalar-addentity-addjoin-parameter-example).

### What is the benefit of native sql query support in hibernate?

Native SQL Query comes handy when we want to execute database specific queries that are not supported by Hibernate API such as query hints or the CONNECT keyword in Oracle Database.

### What is Named SQL Query?

Hibernate provides Named Query that we can define at a central location and use them anywhere in the code. We can created named queries for both HQL and Native SQL.

Hibernate Named Queries can be defined in Hibernate mapping files or through the use of JPA annotations @NamedQuery and @NamedNativeQuery.

### What are the benefits of Named SQL Query?

Hibernate Named Query helps us in grouping queries at a central location rather than letting them scattered all over the code.  
Hibernate Named Query syntax is checked when the hibernate session factory is created, thus making the application fail fast in case of any error in the named queries.  
Hibernate Named Query is global, means once defined it can be used throughout the application.

However one of the major disadvantage of Named query is that it’s hard to debug, because we need to find out the location where it’s defined.

### What is the benefit of Hibernate Criteria API?

Hibernate provides Criteria API that is more object oriented for querying the database and getting results. We can’t use Criteria to run update or delete queries or any DDL statements. It’s only used to fetch the results from the database using more object oriented approach.

Some of the common usage of Criteria API are:

* 1. Criteria API provides Projection that we can use for aggregate functions such as sum(), min(), max() etc.
  2. Criteria API can be used with ProjectionList to fetch selected columns only.
  3. Criteria API can be used for join queries by joining multiple tables, useful methods are createAlias(), setFetchMode() and setProjection()
  4. Criteria API can be used for fetching results with conditions, useful methods are add() where we can add Restrictions.
  5. Criteria API provides addOrder() method that we can use for ordering the results.

Learn some quick examples at [Hibernate Criteria Example](http://www.journaldev.com/2963/hibernate-criteria-example-tutorial).

### How to log hibernate generated sql queries in log files?

We can set below property for hibernate configuration to log SQL queries.

|  |  |
| --- | --- |
| 1 | <property name="hibernate.show\_sql">true</property> |

However we should use it only in Development or Testing environment and turn it off in production environment.

### What is Hibernate Proxy and how it helps in lazy loading?

Hibernate uses proxy object to support lazy loading. Basically when you load data from tables, hibernate doesn’t load all the mapped objects. As soon as you reference a child or lookup object via getter methods, if the linked entity is not in the session cache, then the proxy code will go to the database and load the linked object. It uses javassist to effectively and dynamically generate sub-classed implementations of your entity objects.

### How to implement relationships in hibernate?

We can easily implement one-to-one, one-to-many and many-to-many relationships in hibernate. It can be done using JPA annotations as well as XML based configurations. For better understanding, you should go through following tutorials.

* 1. [Hibernate One to One Mapping](http://www.journaldev.com/2916/hibernate-one-to-one-mapping-annotation-and-xml-configuration-example)
  2. [Hibernate One to Many Mapping](http://www.journaldev.com/2924/hibernate-one-to-many-mapping-annotation-and-xml-configuration-example-tutorial)
  3. [Hibernate Many to Many Mapping](http://www.journaldev.com/2934/hibernate-many-to-many-mapping-join-tables-annotation-and-xml-configuration-example)

### How transaction management works in Hibernate?

Transaction management is very easy in hibernate because most of the operations are not permitted outside of a transaction. So after getting the session from SessionFactory, we can call sessionbeginTransaction() to start the transaction. This method returns the Transaction reference that we can use later on to either commit or rollback the transaction.

Overall hibernate transaction management is better than JDBC transaction management because we don’t need to rely on exceptions for rollback. Any exception thrown by session methods automatically rollback the transaction.

### What is cascading and what are different types of cascading?

When we have relationship between entities, then we need to define how the different operations will affect the other entity. This is done by cascading and there are different types of it.

Here is a simple example of applying cascading between primary and secondary entities.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | import org.hibernate.annotations.Cascade;    @Entity  @Table(name = "EMPLOYEE")  public class Employee {    @OneToOne(mappedBy = "employee")  @Cascade(value = org.hibernate.annotations.CascadeType.ALL)  private Address address;    } |

Note that Hibernate CascadeType enum constants are little bit different from JPAjavax.persistence.CascadeType, so we need to use the Hibernate CascadeType and Cascade annotations for mappings, as shown in above example.  
Commonly used cascading types as defined in CascadeType enum are:

* 1. None: No Cascading, it’s not a type but when we don’t define any cascading then no operations in parent affects the child.
  2. ALL: Cascades save, delete, update, evict, lock, replicate, merge, persist. Basically everything
  3. SAVE\_UPDATE: Cascades save and update, available only in hibernate.
  4. DELETE: Corresponds to the Hibernate native DELETE action, only in hibernate.
  5. DETATCH, MERGE, PERSIST, REFRESH and REMOVE – for similar operations
  6. LOCK: Corresponds to the Hibernate native LOCK action.
  7. REPLICATE: Corresponds to the Hibernate native REPLICATE action.

### How to integrate log4j logging in hibernate application?

Hibernate 4 uses JBoss logging rather than slf4j used in earlier versions. For log4j configuration, we need to follow below steps.

* 1. Add log4j dependencies for maven project, if not maven then add corresponding jar files.
  2. Create log4j.xml configuration file or log4j.properties file and keep it in the classpath. You can keep file name whatever you want because we will load it in next step.
  3. For standalone projects, use static block to configure log4j using DOMConfigurator orPropertyConfigurator. For web applications, you can use ServletContextListener to configure it.

That’s it, our setup is ready. Create org.apache.log4j.Logger instance in the java classes and start logging. For complete example code, you should go through [Hibernate log4j example](http://www.journaldev.com/2984/hibernate-4-log4j-configuration-example) and [Servlet log4j example](http://www.journaldev.com/1997/servlet-example-in-java-with-database-connection-and-log4j-integration).

### How to use application server JNDI DataSource with Hibernate framework?

For web applications, it’s always best to allow servlet container to manage the connection pool. That’s why we define JNDI resource for DataSource and we can use it in the web application. It’s very easy to use in Hibernate, all we need is to remove all the database specific properties and use below property to provide the JNDI DataSource name.

|  |  |
| --- | --- |
| 1 | <property name="hibernate.connection.datasource">java:comp/env/jdbc/MyLocalDB</property> |

For a complete example, go through [Hibernate JNDI DataSource Example](http://www.journaldev.com/2905/hibernate-tomcat-jndi-datasource-example-tutorial).

### How to integrate Hibernate and Spring frameworks?

Spring is one of the most used Java EE Framework and Hibernate is the most popular ORM framework. That’s why Spring Hibernate combination is used a lot in enterprise applications. The best part with using Spring is that it provides out-of-box integration support for Hibernate with **Spring ORM** module. Following steps are required to integrate Spring and Hibernate frameworks together.

* 1. Add hibernate-entitymanager, hibernate-core and spring-orm dependencies.
  2. Create Model classes and corresponding DAO implementations for database operations. Note that DAO classes will use SessionFactory that will be injected by Spring Bean configuration.
  3. If you are using Hibernate 3, you need to configureorg.springframework.orm.hibernate3.LocalSessionFactoryBean ororg.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean in Spring Bean configuration file. For Hibernate 4, there is single classorg.springframework.orm.hibernate4.LocalSessionFactoryBean that should be configured.
  4. Note that we don’t need to use Hibernate Transaction Management, we can leave it to Spring declarative transaction management using @Transactional annotation.

For complete example go through [Spring Hibernate Integration](http://www.journaldev.com/3524/spring-hibernate-integration-example-tutorial-spring-4-hibernate-3-and-hibernate-4) and [Spring MVC Hibernate Integration](http://www.journaldev.com/3531/spring-mvc-hibernate-mysql-integration-crud-example-tutorial).

### What is HibernateTemplate class?

When Spring and Hibernate integration started, Spring ORM provided two helper classes –HibernateDaoSupport and HibernateTemplate. The reason to use them was to get the Session from Hibernate and get the benefit of Spring transaction management. However from Hibernate 3.0.1, we can use SessionFactory getCurrentSession() method to get the current session and use it to get the spring transaction management benefits. If you go through above examples, you will see how easy it is and that’s why we should not use these classes anymore.

One other benefit of HibernateTemplate was exception translation but that can be achieved easily by using @Repository annotation with service classes, shown in above spring mvc example. This is a trick question to judge your knowledge and whether you are aware of recent developments or not.

### How to integrate Hibernate with Servlet or Struts2 web applications?

Hibernate integration with Servlet or Struts2 needs to be done using ServletContextListener, a complete example can be found at [Hibernate Struts2 Integration Example](http://www.journaldev.com/3557/struts2-hibernate-integration-example-tutorial).

### Which design patterns are used in Hibernate framework?

Some of the design patterns used in Hibernate Framework are:

* 1. Domain Model Pattern – An object model of the domain that incorporates both behavior and data.
  2. Data Mapper – A layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself.
  3. [Proxy Pattern](http://www.journaldev.com/1572/proxy-design-pattern-in-java-example-tutorial) for lazy loading
  4. [Factory pattern](http://www.journaldev.com/1392/factory-design-pattern-in-java) in SessionFactory

### What is Hibernate Validator Framework?

Data validation is integral part of any application. You will find data validation at presentation layer with the use of Javascript, then at the server side code before processing it. Also data validation occurs before persisting it, to make sure it follows the correct format.

Validation is a cross cutting task, so we should try to keep it apart from our business logic. That’s why JSR303 and JSR349 provides specification for validating a bean by using annotations. Hibernate Validator provides the reference implementation of both these bean validation specs. Read more at[Hibernate Validation Example](http://www.journaldev.com/3626/hibernate-validator-jsr303-example-tutorial).

### What are best practices to follow with Hibernate framework?

Some of the best practices to follow in Hibernate are:

* 1. Always check the primary key field access, if it’s generated at the database layer then you should not have a setter for this.
  2. By default hibernate set the field values directly, without using setters. So if you want hibernate to use setters, then make sure proper access is defined as @Access(value=AccessType.PROPERTY).
  3. If access type is property, make sure annotations are used with getter methods and not setter methods. Avoid mixing of using annotations on both filed and getter methods.
  4. Use native sql query only when it can’t be done using HQL, such as using database specific feature.
  5. If you have to sort the collection, use ordered list rather than sorting it using Collection API.
  6. Use named queries wisely, keep it at a single place for easy debugging. Use them for commonly used queries only. For entity specific query, you can keep them in the entity bean itself.
  7. For web applications, always try to use JNDI DataSource rather than configuring to create connection in hibernate.
  8. Avoid Many-to-Many relationships, it can be easily implemented using bidirectional One-to-Many and Many-to-One relationships.
  9. For collections, try to use Lists, maps and sets. Avoid array because you don’t get benefit of lazy loading.
  10. Do not treat exceptions as recoverable, roll back the Transaction and close the Session. If you do not do this, Hibernate cannot guarantee that in-memory state accurately represents the persistent state.
  11. Prefer DAO pattern for exposing the different methods that can be used with entity bean
  12. Prefer lazy fetching for associations