**What is Hibernate?**  
Hibernate is an ORM (Object-relational Mapping) framework, which allows the developer to concentrate on business logic by taking care of persistence of data by itself. Java developer can write code using object and Hibernate can take care of creating those object from data loaded from the database and saving update back to the database.  
  
**What are the advantages of Hibernate over JDBC?**   
Apart from Persistence i.e. saving and loading data from Database, Hibernate also provides following benefits  
1) Caching – better performance  
2) Lazy Loading  
3) Relationship management and provides code for mapping an object to the data  
4) The developer is free from writing code to load/store data into the database (less boilerplate code)

5) Implicit transaction management

6) HQL over SQL because is more Object Oriented

7) Hibernate provides an option for tables creation

8) Hibernate supports JPA specification, can be easily replaced

**JPA and Hibernate**

JPA is just a specification, meaning there is no implementation. You can annotate your classes as much as you would like with JPA annotations, however without an implementation nothing will happen. Think of JPA as the guidelines that must be followed or an interface, while Hibernate's JPA implementation is code that meets the API as defined by the JPA specification and provides the under the hood functionality.

When you use Hibernate with JPA you are actually using the Hibernate JPA implementation. The benefit of this is that you can swap out Hibernate's implementation of JPA for another implementation of the JPA specification. When you use straight Hibernate you are locking into the implementation because other ORMs may use different methods/configurations and annotations, therefore you cannot just switch over to another ORM.

**Hibernate SessionFactory**

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.mSessionFactory also provide methods to get the Class metadata and Statistics instance to get the stats of query executions, second level cache details etc.Internal state of SessionFactory is immutable, so it’s thread safe. Multiple threads can access it simultaneously to get Session instances.

**Hibernate Session**

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

**Difference between get() vs load() method in Hibernate?** 

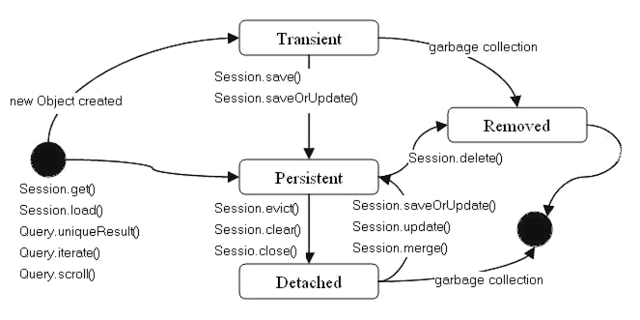
1. get() loads the data as soon as it’s called whereas load() returns a proxy object and loads data only when it’s actually required, so load() is better because it support lazy loading.
2. Since load() throws exception when data is not found, we should use it only when we know data exists.
3. We should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

**What is N+1 SELECT problem in Hibernate?**(detailed answer)  
The N+1 SELECT problem is a result of lazy loading and load on demand fetching strategy. In this case, Hibernate ends up executing N+1 SQL queries to populate a collection of N elements. For example, if you have a List of N Items where each Item has a dependency on a collection of Bid object. Now if you want to find the highest bid for each item then Hibernate will fire 1 query to load all items and N subsequent queries to load Bid for each item. So in order to find the highest bid for each item your application end up firing N+1 queries.  It's one of the important Hibernate interview questions and I suggest to read chapter 13 of [Java Persistence with Hibernate](http://www.amazon.com/Java-Persistence-Hibernate-Christian-Bauer/dp/1932394885?tag=javamysqlanta-20) to understand this problem in more details.  
  
**What are some strategies to solve the N+1 SELECT problem in Hibernate?**(detailed answer)  
This is the follow-up question of previous Hibernate interview question. If you answer the last query correctly then you would be most likely asked this one. Here are some strategies to solve the N+1 problem:  
1) pre-fetching in batches, this will reduce N+1 problem to N/K + 1 problem where  K is size of batch  
2) subselect fetching strategy  
3) disabling lazy loading  
  
  
**What is the difference between save() and persist() method in Hibernate?**   
Main difference between save() and persist() method is that, save returns a Serializable object while return type of persist() method is void, so it doesn't return anything.   
  
**What is the requirement for a Java object to become Hibernate entity object?**   
It should not be final and must provide a default, no-argument constructor.

**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().

[](https://4.bp.blogspot.com/-0FvMOmnE7ec/Vrm6gpn-yvI/AAAAAAAAEvA/fb9ZfeJL9Uw/s1600/Difference+between+save+vs+persist+in+Hibernate.png)

**What are different types of caches available in Hibernate?**   
This is another common Hibernate interview question. Hibernate provides the out-of-box caching solution but there are many caches e.g. first level cache, second level cache and query cache. First level cache is maintained at Session level and cannot be disabled but the second level cache is required to be configured with external cache provider like EhCache.  
  
**What is the difference between first and second level cache in Hibernate?**   
The first level cache is maintained at Session level while the second level cache is maintained at SessionFactory level and is shared by all sessions.   
  
**What is different between Session and Sessionfactory in Hibernate?**(detailed answer)  
The main difference between Session and SessionFactory is that former is a single-threaded, short-lived object while later is Immutable and shared by all Session. It also lives until the Hibernate is running. Another difference between Session and SessionFactory is that former provides first level cache while SessionFactory provides the Second level cache.  
  
**What is criterion query in hibernate?**(detailed answer)  
Criteria is a simplified API for retrieving entities by composing Criterion objects also known as Criterion query. This is a very convenient approach for functionality like "search" screens where you can filter data on multiple conditions as shown in the following example:

List books = session.createCriteria(Book.class)

.add(Restrictions.like("name", "java%") )

.add(Restrictions.like("published\_year", "2015"))

.addOrder(Order.asc("name") )

.list();

**What are other ORM frameworks? Any alternative of Hibernate?**  
Apache OpenJPA,

Ebean – used by Play Framework

TopLink from Oracle

**What is the difference between save() and saveOrUpdate() method of Hibernate?**([detailed answer](http://javarevisited.blogspot.com/2012/09/difference-hibernate-save-vs-persist-and-saveOrUpdate.html))  
Though both save() and saveOrUpdate() method is used to store object into Database, the key difference between them is that save can only INSERT records but saveOrUpdate() can either INSERT or UPDATE records.  
  
  
**What is difference between getCurrentSession() and openSession() in Hibernate?**(detailed answer)

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.

<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.  
  
**What is Hibernate Query Language (HQL)?**   
Hibernate query language, HQL is an object-oriented extension to SQL. It allows you to query, store, update, and retrieve objects from a database without using SQL. This question is also similar to the earlier question about Criterion query, Java developers who have not used Hibernate extensively will not know much about features like HQL and Criterion.  
  
**Hibernate saving objects methods.**

save() – used to save entity into database. It can be invoked outside of a transaction, but then only only the primary entity gets saved without cascading, unless we flush the session. Not recommended to be used because of that – data inconsistency.

persist() – it is similar to save() but with transaction, and it adds the entity object to the **persisten context**, so any further changes are tracked. As it needs to be used only within the boundary of a transaction it is safe and takes care of any cascaded obects.

saveOrUpdate() – results into insert or update based on the provided data. It also can be used without transaction with the same consequences as save() method. It adds the entity object to persistent context.

update() – should be used when we know that we are only updating the entity information. This operation adds the entity object to persistent context

merge() – can be used to update existing values. However, this method creates a copy from the passed entity object and returns it. The returned object is part of the persistent context, passed object is not.

**The difference between sorted and ordered collection in Hibernate?**

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/comparable-and-comparator-in-java-example).

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.

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

Hibernate uses Reflection API 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.

**How do you log SQL queries issued by the Hibernate framework in Java application?**  
You can use the show\_sql property to log SQL queries issued by the Hibernate framework, Just add the following line in your Hibernate configuration file:

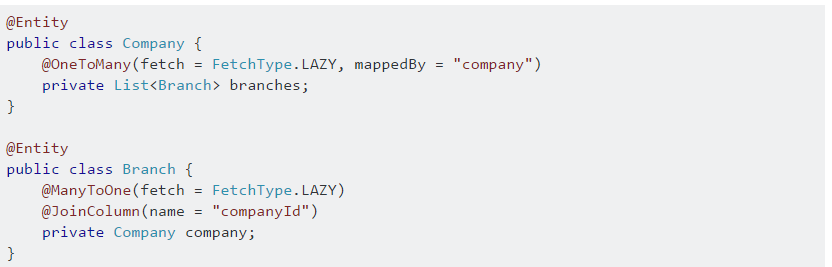
<property name=”show\_sql”> true </property>

**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.

**JoinColumn and MappedBy**

The annotation @JoinColumn indicates that this entity is the owner of the relationship (that is: the corresponding table has a column with a foreign key to the referenced table), whereas the attribute mappedBy indicates that the entity in this side is the inverse of the relationship, and the owner resides in the "other" entity. This also means that you can access the other table from the class which you've annotated with "mappedBy" (fully bidirectional relationship).



Usually when you link 2 tables in any DB System, just 1 of those has the foreign key constraint to the other one right?

**What MappedBy allows you to link from the table not containing the constraint to the other table.**

About @JoinColumn it's easier, here you have in table foreign key, so you tell hibernate this is not only a column, but a column that must **join** a table.

**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.

**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.

<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.

Query query = session.createQuery("from Employee");

query.setCacheable(true);

query.setCacheRegion("ALL\_EMP");

**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.

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 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 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

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

**How to integrate Hibernate and Spring frameworks?**

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.

* Add hibernate-entitymanager, hibernate-core and spring-orm dependencies.
* 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.
* 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.
* Note that we don’t need to use Hibernate Transaction Management, we can leave it to Spring declarative transaction management using @Transactional annotation.

**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 are best practices to follow with Hibernate framework?**

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

**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.