**Q. What is Hibernate?  
Ans**. Hibernate is an ORM (Object-relational Mapping) framework, which allows the developer to concentrate on business logic by taking care of the 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.  
  
**Q. What are the advantages of Hibernate over JDBC?**  
Apart from Persistence i.e. saving and loading data from Database, Hibernate also provides the following benefits  
1) Caching  
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.  
  
**Q. Difference between first level and second level cache in Hibernate**

**Ans**. The main difference between first level and second level cache in Hibernate is that the **first level is maintained at the Session level** and **accessible only to the Session**, while the **second level cache is maintained at the SessionFactory level and available to all Sessions**. This means, we can use the first level cache to store local data, i.e. the data which is needed by the Session, and we can use the second-level cache to store global data, i.e. something which can be shared across sessions.   
Caching is one of the powerful features of Hibernate and probably one of the most substantial reasons to use the Hibernate framework. It allows developers to build a more responsive web application by **minimizing the number of database transactions**.  
Hibernate maintains different caches for the different purpose, e.g. first level cache at Session level, Second level cache at the SessionFactory level and Query Cache to cache query and its results .

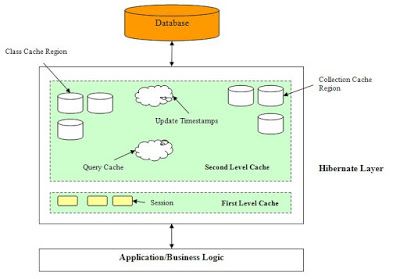
**Q .What is First Level Cache in Hibernate?**

**Ans.** As the name suggests, the first-level cache is **the first cache hibernate consults before loading an object from the database**. It is maintained at Session level, and it's by default enabled.  
If we know, hibernate then we know that Session is the interface between Hibernate and Database. **We load objects using Session, e.g. by calling get() or load() method or by executing queries.**  
When we ask load method to return a Person (an object stored in database) with Id=1, **first time it's loaded from database, that too lazily**, when any method of that object is called other than getId(), **next time if we load the same object then Hibernate doesn't go to Database,** **instead it returns the object from first level cache maintained at Session level.**  
Similarly, when we update a particular object, Hibernate defer the database call to combine multiple database transactions into one, this way, Hibernate improves the performance of our web application.  
**The data in the first level cache is maintained as long as Session is open, as soon as we close the Session, all data is lost**. Next time even if we load the same object, e.g. **Person with the same id, Hibernate will again go to Database to load that object**, provided Second-level Cache is disabled .

Caching at Session level has some memory implication, especially if we are loading lots of large objects. Long-lived sessions with several large objects will take more memory and can cause out of memory errors into our web application.

**Q. What is Second-Level Cache in Hibernate?**

This is an optional Cache which Hibernate provides. Unlike first level cache which is accessible only to the session who maintains it, Second-level Cache is accessible to all Sessions.  
This means **if one Session loads an object, like Person with id=1**and **Second session also loads the same object**, **only one database call will be made.** **Next session will get the data from the Second-level Cache.**  
  
The Second Level cache is by default disabled. Hibernate also does not provide any caching implementation for that. Instead, it gives CacheProvider interface and any third party Cache which implements CacheProvider interface can be hooked as Second level cache, e.g. **EHCache or NCache.**  
We also need to define Concurrency Strategy to be used along with Second level cache, e.g. Transactions, read-write, nonstrict-read-write, or read-only..



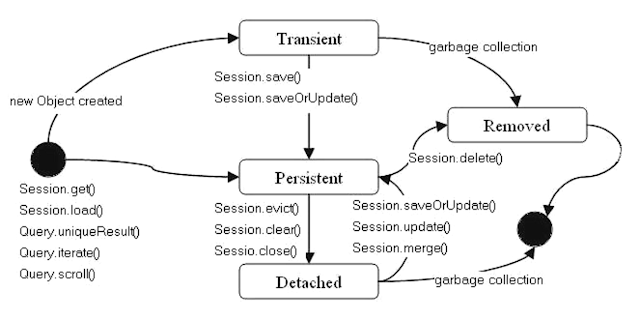
The second-level cache can improve the performance of our Java Web application even further, but this should only be used to make a useful application better and should not be used to address performance problem faced by the application.  
  
  
**First level Cache vs. Second-level Cache in Hibernate**

Now that we have got some basic understanding of the first level and second level cache, here are some differences between them:  
1) The primary difference is that first level cache is maintained at the Session level while the second level cache is maintained at SessionFactory level.  
2)  The data stored in the **first level cache is accessible to the only** **Session** who maintains it, while the **second level cache is accessible to all.**  
3) The **first level cache is by default enabled** while the **second level cache is by default disabled.**  
A couple of things to know about hibernate first level cache:  
1) We can use **Session.evict() to remove the loaded entity from the first level cache**, can use refresh() method to refresh the cache and can use clear() method to remove all entities in cache.  
2) **We cannot disable the first level cache, it always enabled**.  
3) Hibernate entities or database rows remain in cache only until Session is open, once Session is closed, all associated cached data is lost.  
  
That's all about the difference between 1st and 2nd level cache in Hibernate. Just remember that first level cache is local to Session object and cannot be shared between multiple sessions, but the second-level cache is maintained at the SessionFactory level and shared among all sessions in Hibernate.   
  
**Q. 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.  
The 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.**  
  
**Q. Does Hibernate Session interface thread-safe in Java?    
Ans. No**, Session object is not thread-safe in Hibernate and intended to be used with-in single thread in the application.  
  
**Q.Difference between save(), saveOrUpdate() and persist() in Hibernate Session**

**Ans**. Session interface in Hibernate provides a couple of methods to move an object from new or transient state to persistent state e.g. save(), saveOrUpdate() and persist() are used to store an object into the database, but there are some significant differences between them. The **Session.save() method** does an INSERT to store the object into the database and it also **return the identifier generated by the database**. On the other hand**, saveOrUpdate()**can be used to reattach a detached object in Hibernate Session i.e. **saveOrUpdate** **can do INSERT or UPDATE depending upon whether object exists in database or not.** The third method, **persist()** is similar to Session.save() i.e. it is used to move a transient object to the **persistent state by storing it into the database but it doesn't return the database identifier.** This was the basic difference between save() vs saveOrUpdate() vs persist() method and suggest when to use which method.   
**Session.save() vs Session.saveOrUpdate() vs Session.Persist**

As I said, all three methods save(), persist() and saveOrUpdate belongs to Hibernate's Session interface. They all are used to bring a new object into persistence context or a detached object, which was associated with a Hibernate Session in past but currently it's not associated into active persistence context.

**1**) The main difference between save() and saveOrUpdate() method is that **save() method performs an INSERT operation** to store the object into the database, **but INSERT will fail if the primary key is already persistent** i.e. object already exists in the database. This is why, we should only call **save() with an absolutely new object which doesn't have any database identifier. Calling save() with the detached object will fail**. This is opposite of **saveOrUpdate() method, which can do either INSERT or UPDATE SQL query depending upon whether an object exists in the database or not**. The *saveOrUpdate()* method **first executes a SELECT query**to determine if it needs to do an INSERT or UPDATE operation.**2**) Another key difference between save() and saveOrUpdate() method is that **save()** is used to **bring a transient object to persistent** state **but saveOurUpdate() can bring both transient (new) and detached (existing) object into persistent state**. It is often used to re-attach a detached object into Session.  
  
Here is an object's lifecycle and state transition diagram in Hibernate which shows that save() and saveOrUpdate() can move an object from transient to persistent state.



**3**) Coming to persist() method, the difference between save() and persist() method is that save() **returns the generated database identifier**, a Serializable object but **persist()** method doesn't return anything. **It's return type is void**.  
  
For example:  
System.out.println(session.**save**(aCoin));

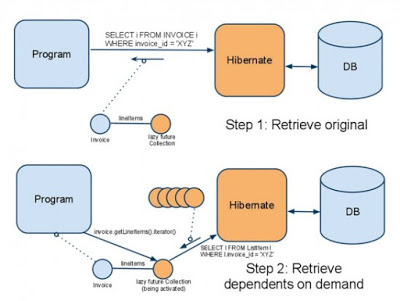
will print the generated primary key, but the following line **will throw a compile time error** because persist()'s return type is **void**.  
**System.out.println(session.persist(aCoin));**

Use the save() method to store new object into database and when we need the generated database identifier, otherwise use persist() method. We can use saveOrUpdate() to reattach a detached object into Hibernate Session.

**Q. Why JPA Entity or Hibernate Persistence Class Should Not be Final?**

**Ans**: - The use of proxies is the core feature of Hibernate implementing key **performance** features e.g**. lazy loading and lazy associations fetching**. In order to use a proxy in place of real class, our hibernate persistence class must be either non-final or the implementation of an interface that declares all of the public methods. Why? because we cannot extend a final class in Java, and to stand up as a proxy, the **proxy class must satisfy the IS-A relation,** which comes either by extending a class using "extends", or implementing an interface using "implements".  
  
By the way, it doesn't mean that we cannot persist our final entity class, we can, but this will limit Hibernate's ability to use proxies for lazy association fetching, which will affect the performance of Java application,   
Hibernate framework makes extensive use or proxy e.g**. it uses proxies when we call the load() method**, **it also uses proxies for lazily loading associations** and collections of an entity class. Even Hibernate  Reference Manual says **"Prefer non-final classes".**Not using proxies is bad for performance because more SQL than we'd like maybe fired against the database, which also increases database round-trips. We can verify this by printing the class name of our entity class and look at Hibernate query logs, as shown in the following example:  
Let's say we load our entity class e.g. a **Book**, an **Employee** or a **Customer** using the load() method and further print name of the class, we will see a different name e.g.  
Customer customer = (Customer) session.load(Customer.class, 1L);

logger.info("Name of Customer Entity class : " + customer.getClass());

will print "sample.hibernate.Customer\_$$\_javassist\_0" which is a JavaAssist generated the proxy.  
**We will not see any select query** fired **because the load is lazy** and **it will not load object until** **we call any method other than the getId(),**  
Now, if we make the class final and **reprint** the name of the class, we will see the actual name of the class as "Customer". We will also see the select queries fired by hibernate to initialize the object.  
We should now understand how costly it can be if we are creating too many objects and only accessing their getId() method, as it will result in lots of database calls. That's why **we should** **not make our JPA Entity or Hibernate persistence class final in Java**. 

**Important points to about Entity class, Final, and Proxying**

**1) Hibernate does not create a proxy for the final class**, instead, they use the real class, but **Hibernate does create a proxy for a non-final class with final methods**.  
**2**) Since in Java, we cannot override final methods, **the code inside the final method remains unchanged in the proxy class.** This means, **if we call a final method on a proxy, it will simply delegate the call to the superclass method.** **We should be careful not to modify state on final method because calling them on proxy has a good chance of throwing NullPointerException,** as the object was likely to be not initialized at that point.  
In short**, avoid final methods on hibernate entity class,** until we know what we are doing. Hibernate reference also stresses this point "**We should avoid declaring public final methods as this will again limit the ability to generate proxies from this class.** If we want to use a class with public final methods, we must explicitly disable proxying".  
**3**) As per Hibernate documentation, **if we're going to make a class final we should explicitly disable proxy generation by adding @Proxy(lazy=false),** but I haven't noticed any differences between doing that and just making the class final.  
**4**) We should also disable proxy generation if we're going to make any public methods final in persistent class, as hibernate will not be able to override them to put its smart code which triggers lazy loading, etc.  
**5**) If we really want to make the Hibernate entity classes final then **we can do it by having our entity implement an interface that declares all of its public methods. This will still allow Hibernate to use proxies**.  
In short, making a JPA Entity or Hibernate Persistence class final, limits the ability of Hibernate to use Proxies, which in turn prevent Hibernate from applying some performance optimizations. Without proxies, our application loses lazy loading, and lazy association fetching will issue more SQL queries and make more database roundtrip, which will cost performance.

**Q. What is different between Session and SessionFactory in Hibernate**

**Ans**. The main difference between Session and SessionFactory is that the **Session** is a single-threaded, short-lived object while **SessionFactory** is Immutable and shared by all Session.  
SessionFactory also lives until the Hibernate is running. Another difference between Session and SessionFactory is that Session provides first level cache while SessionFactory provides the Second level cache.  
**Q. What is criterion query in hibernate?   
Ans**. 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 we 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();

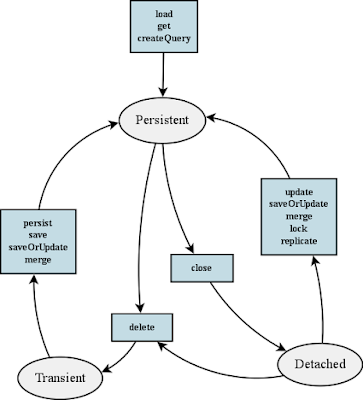
**Q. What are other ORM frameworks? Any alternative of Hibernate?  
Ans** .This is a general question, sometimes asked to start the conversation and other times to finish the interview. **EJB** and **TopLink** from Oracle are two of the most popular alternative to Hibernate framework.  
  
**Q. What is difference between getCurrentSession() and openSession() in Hibernate?    
Ans**. Used both getCurrentSession() and openSession() to obtain an instance of the Session object.

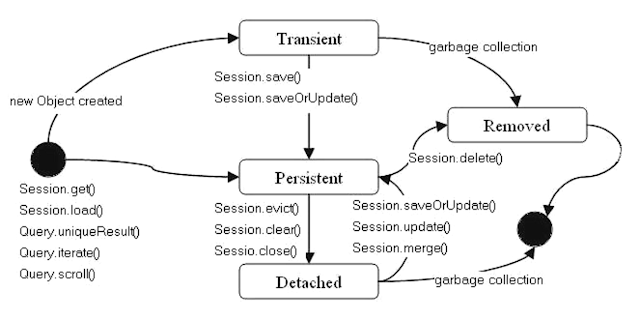
**Q. When do we use merge() and update() in Hibernate?  
Ans**. We should **use update() if we are sure that the Hibernate session does not contain an already persistent instance with the same id** and **use merge() if we want to merge our modifications at any time without considering the state of the session.**

**Q. The difference between sorted and ordered collection in Hibernate?**  
**Ans**. The main difference between sorted and ordered collection is that **sorted** collection sort the data in JVM's heap memory using Java's collection framework sorting methods while the **ordered** collection is sorted using order by clause in the database itself.  
A sorted collection is more suited for small dataset but for a large dataset, it's better to use ordered collection to avoid OutOfMemoryError in Java application.

**Q. How do we log SQL queries issued by the Hibernate framework in Java application?**We can use the show\_sql property to log SQL queries issued by the Hibernate framework, Just add the following line in our Hibernate configuration file:  
**<property name=”show\_sql”> true </property>**

**Q. What are the three states of a Hibernate Persistence object can be?**  
The Hibernate persistent or entity object can live in the following three states:  
1) transient  
2) persistent  
3) detached  
  
**Q. What is the difference between the transient, persistent and detached state in Hibernate?  
Ans.** **New objects created in Java program but not associated with any hibernate Session** are said to be in the **transient** state.  
On the other hand, **an object which is associated with a Hibernate session is called Persistent object.** **While an object which was earlier associated with Hibernate session but currently it's not associate is known as a detached object**.  
We can call **save() or persist()** method to store those object into the database and **bring them into the Persistent state**. Similarly, we can **re-attach a detached object** to hibernate sessions by calling either **update() or saveOrUpdate()** method.

**Q. Which cache is used by Session Object in Hibernate? First level or second level cache?**  
**A Session object uses the first-level cache**. As I told before the second level cache is used at SessionFactory level. This is a good question to check if the Candidate has been working in hibernate or not. If he has not worked in Hibernate from a long time then he would get confused in this question.  
  
**Q. What is the difference between save and saveOrUpdate or Difference**   




**Q. What is the difference between get and load in Hibernate?**

Ans**.** Main difference between get and load is that, get will hit the database if object is not found in the cache and **returned completely initialized object**, which **may involve several database call** while **load() method can return proxy if the object is not found in the cache** and **only hit database if any method other than getId()** is called. This can save a lot of performance in some cases.

**Q. What is named SQL query in Hibernate?**

**Ans**. Named queries are SQL queries which are defined in mapping document using <sql-query> tag and called using Session.getNamedQuery() method.   
Named query allows we to refer a particular query by the name we provided, by the way, we can define named query in hibernate either by using annotations or XML mapping file, as I said above. @**NameQuery** is used to define a single named query and @**NameQueries** is used to define multiple named query in hibernate.

**Q. What is SessionFactory in Hibernate? is SessionFactory thread-safe?**

**Ans. SessionFactory**, as the name suggests, is a factory to hibernate Session objects. **SessionFactory** is often built during start-up and used by application code to get the session object. It acts as a single data store and **it is also thread-safe so that multiple threads can use the same SessionFactory.**   
Usually, a Java JEE application has just one SessionFactory, and individual threads, which are servicing client’s requests obtain hibernate Session instances from this factory, that’s why any implementation of SessionFactory interface must be thread-safe.  
  
Also, the internal state of SessionFactory, which contains all metadata about Object/Relational mapping is Immutable and can not be changed once created.

**Q. What is Session in Hibernate? Can we share a single Session among multiple threads in Hibernate?**

**Ans**. Session represents a small unit of work in Hibernate, they **maintain a connection with the database** and they are **not thread-safe**, it means we cannot share Hibernate Session between multiple threads. Though **Session obtains database connection lazily** it **is good to close the session** as soon as we are done with it.

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

**Ans**.. A **sorted collection** is **sorted in memory by using Java Comparator** while an **ordered** **collection uses the database's order** by clause for ordering. For large data set it's better to use ordered collection to avoid any OutOfMemoryError in Java, by trying to sort them in memory.

**Q. What does Session lock() method do in Hibernate?**

**Ans**. **Session's lock() method reattach objects without synchronizing or updating with the database**. So we need to be very careful while using the lock() method. By the way, **we can always use the Session's update() method to sync with the database during attachment.**  
Sometimes this Hibernate question is also asked as what is the difference between Session's lock() and update() method.

**Q. What is the query cache in Hibernate?**

**Ans**.  **QueryCache actually stores the result of SQL query for future calls.** Query cache can be used along with second-level cache for improved performance. Hibernate support various open-source caching solution to implement Query cache e.g. **EhCache**.

**Q. Why it's important to provide no-argument constructor in Hibernate Entities?**

**Ans**. Every Hibernate Entity class **must contain a no-argument constructor,** because **Hibernate framework creates an instance of them using Reflection API**, by calling Class.newInstance() method. This method will throw InstantiationException if it doesn't found any argument constructor inside Entity class.

**Q. What are some strategies to solve the N+1 SELECT problem in Hibernate?**  
**Ans**. Here are some strategies to solve the N+1 problem:  
1) pre-fetching in batches, this will reduce the N+1 problem to N/K + 1 problem where  K is the size of the batch  
2) subselect fetching strategy  
3) disabling lazy loading