JPA Annotations for mapping java object to database table

- javax.persistence.Entity: Specifies that the class is an entity. This annotation can be applied on Class, Interface of Enums.

```
import javax.persistence.Entity;

@Entity
public class Employee implements Serializable {
}
```

- @Table: It specifies the table in the database with which this entity is mapped. In the example below the data will be stores in the “employee” table. Name attribute of @Table annotation is used to specify the table name.

```
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name = "employee")
public class Employee implements Serializable {
}
```

- @Column: Specify the column mapping using @Column annotation. Name attribute of this annotation is used for specifying the table’s column name.

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name = "employee")
public class Employee implements Serializable {

  @Column(name = "employee_name")
  private String employeeName;
}
```

- @Id: This annotation specifies the primary key of the entity.

```
import javax.persistence.*;

@Entity
@Table(name = "employee")
public class Employee implements Serializable {
  @Id
```

```
  @Column(name = "id")
  private int id;
}
```

- @GeneratedValue: This annotation specifies the generation strategies for the values of primary keys.

```
import javax.persistence.*;

@Entity
@Table(name = "employee")
public class Employee implements Serializable {

  @Id
  @Column(name = "id")
  @GeneratedValue(strategy=SEQUENCE, generator="ID_SEQ")
  private int id;
}
```

- @Version: We can control versioning or concurrency using this annotation.

```
import javax.persistence.*;

@Entity
@Table(name = "employee")
public class Employee implements Serializable {
  @Version
  @Column(name = "version")
  private Date version;
}
```

- @OrderBy: Sort your data using @OrderBy annotation. In example below, it will sort all employees_address by their id in ascending order.

```
@OrderBy("id asc")
private Set employee_address;
```

- @Transient: Every non static and non-transient property of an entity is considered persistent, unless you annotate it as @Transient.

```
@Transient
Private int employeePhone;
```

- @Lob: Large objects are declared with @Lob.

```
@Lob
public String getEmployeeAddress() {
   return employeeAddress;
}
```

The above set of annotation are most commonly used JPA annotations to define an entity.

Hibernate Annotations for Mapping between tables

We have another set of annotations that are used to specify the association mapping between different tables and entities.

We will take an example considering the below mentioned scenario.

- Tables 'employee' and 'employeeDetail' have one-to-one association and they share the same primary key.
- Tables 'communication' and 'communicationDetail' are linked by a foreign key. It is also a one to one association.
- Tables 'communication' and 'employee' are linked using a foreign key in many-to-one association with communication being the owner.
- Tables 'employee' and 'employeeStatus' are linked through a foreign key in many-to-one association with employee being the owner.

@OneToOne

Employee and EmployeeDetail entities share the same primary key and we can associate them using @OneToOne and @PrimaryKeyJoinColumn.

In this case the id property of EmployeeDetail is not annotated with @GeneratedValue. The id value of Employee will be used for used for id of EmployeeDetail.

```
@Entity
@Table(name = "employee")
public class Employee implements Serializable {

 @Id
 @Column(name = "id")
 @GeneratedValue
 private int id;

 @OneToOne(cascade = CascadeType.MERGE)
 @PrimaryKeyJoinColumn
 private EmployeeDetail employeeDetail;
}

@Entity
@Table(name = "employeeDetail")
public class EmployeeDetail implements Serializable {

 @Id
 @Column(name = "id")
 private int id;
}
```

Points to note:

- @PrimaryKeyJoinColumn should be used for associated entities sharing the same primary key.
- @JoinColumn & @OneToOne should be mappedBy attribute when foreign key is held by one of the entities.

Communication and CommunicationDetail are linked through a foreign key, so @OneToOne and @JoinColumn annotations can be used. In snippet mentioned below, the id genereated for Communication will be mapped to 'communication_id' column of CommunicationDetail table. @MapsId is used for the same.

```
@Entity
@Table(name = "communicationDetail")
public class CommunicationDetail implements Serializable {

  @Id
  @Column(name = "id")
  @GeneratedValue
  private int id;

  @OneToOne
  @MapsId
  @JoinColumn(name = "communicationId")
  private Communication communication;
}

@Entity
@Table(name = "communication")
public class Communication implements Serializable {

  @Id
  @Column(name = "ID")
  @GeneratedValue
  private Integer id;

  @OneToOne(mappedBy = "communication", cascade = CascadeType.ALL)
  private CommunicationDetail communicationDetail;
}
```

@ManyToOne

Many employees can share the same status. So, employee to employeeStatus is a many to one relation. @ManyToOne annotation can be used for the same.

```
@Entity
@Table(name = "employee")
public class Employee implements Serializable {
```

```
  @ManyToOne
  @JoinColumn(name = "statusId")
  private EmployeeStatus status;
}
```

@OneToMany

Employee to Communication will be a one-to-many relationship. The owner of this relationship is Communication so, we will use 'mappedBy' attribute in Employee to make it bi-directional relationship.

```
@Entity
@Table(name = "employee")
public class Employee implements Serializable {

  @OneToMany(mappedBy = "employee", fetch = FetchType.EAGER)
  @OrderBy("firstName asc")
  private Set communications;
}
```

@PrimaryKeyJoinColumn

This annotation is used to associate entities sharing the same primary key.

```
@Entity
@Table(name = "employee")
public class Employee implements Serializable {

  @Id
  @Column(name = "id")
  @GeneratedValue
  private int id;

  @OneToOne(cascade = CascadeType.MERGE)
  @PrimaryKeyJoinColumn
  private EmployeeDetail employeeDetail;
}
```

@JoinColumn

@JoinColumn annotation is used for one-to-one or many-to-one associations when foreign key is held by one of the entities.

```
@ManyToOne
@JoinColumn(name = "statusId")
private EmployeeStatus status;
```

@JoinTable: @JoinTable and mappedBy should be used for entities linked through an association table.

@MapsId: Two entities with shared key can be persisted using @MapsId annotation.

```
@OneToOne
@MapsId
@JoinColumn(name = "communicationId")
private Communication communication;
```

Hibernate Annotations for inheritance mapping

Now let us try to understand the inheritance mapping annotation in Hibernate.

Hibernate supports the three basic inheritance mapping strategies:

- table per class hierarchy
- table per subclass
- table per concrete class

we will consider example for each type.

1. Table per class hierarchy – single table per Class Hierarchy Strategy.

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="cartype", discriminatorType=DiscriminatorType.STRING )

@DiscriminatorValue("Car")
public class Car { }

@Entity
@DiscriminatorValue("BMW")
public class BMW extends Car { }
```

13. Table per class/subclass – joined subclass Strategy.

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Ship implements Serializable {}

@Entity
@PrimaryKeyJoinColumn
public class Titanic extends Ship {}
```

22. Table per concrete class.

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Aeroplane implements Serializable {}
```

27. @DiscriminatorColumn: As the name suggests this column is the discriminator and this annotation specifies the discriminator column for the SINGLE_TABLE and JOINED Inheritance mapping strategies.

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="cartype", discriminatorType=DiscriminatorType.STRING )
```

That's all for JPA and Hibernate annotations.

Hibernate Interview Questions

---

1) What is hibernate?

Hibernate is an open-source and lightweight ORM tool that is used to store, manipulate and retrieve data from the database.

---

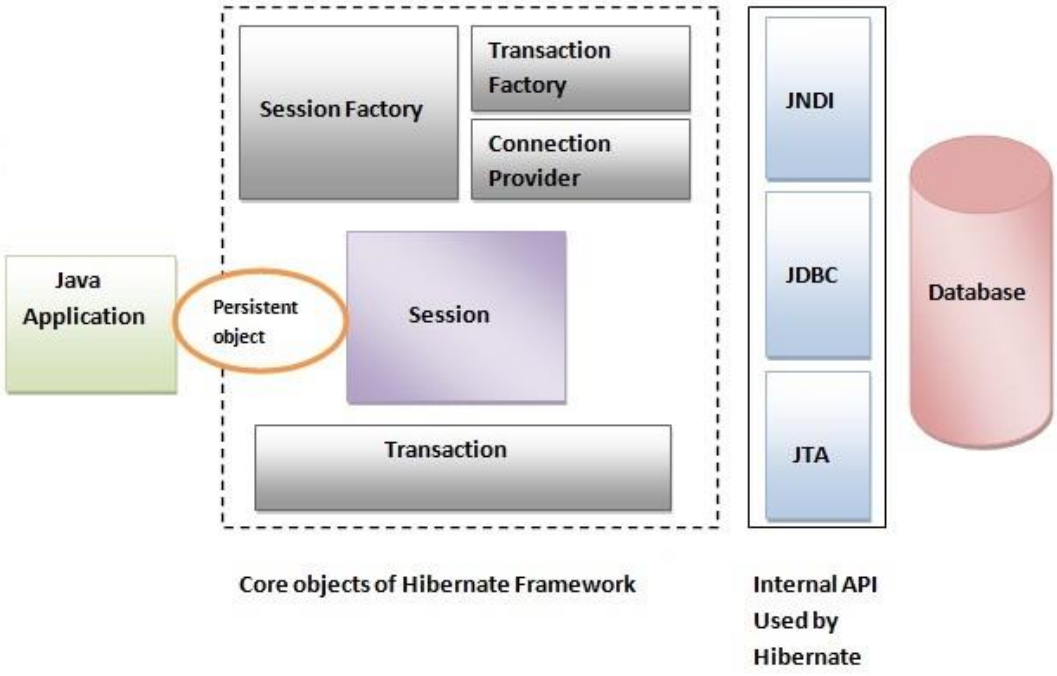
2) What is ORM?

ORM is an acronym for Object/Relational mapping. It is a programming strategy to map object with the data stored in the database. It simplifies data creation, data manipulation and data access.

---

3) Explain hibernate architecture?

Hibernate architecture comprises of many interfaces such as Configuration, SessionFactory, Session, Transaction etc.



4) What are the core interfaces of Hibernate?

The core interfaces of Hibernate framework are:

- Configuration
- SessionFactory
- Session
- Query
- Criteria
- Transaction

---

5) What is SessionFactory?

SessionFactory provides the instance of Session. It is a factory of Session. It holds the data of second level cache that is not enabled by default.

6) Is SessionFactory a thread-safe object?

Yes, SessionFactory is a thread-safe object, many threads cannot access it simultaneously.

7) What is Session?

It maintains a connection between hibernate application and database.
It provides methods to store, update, delete or fetch data from the database such as persist(), update(), delete(), load(), get() etc.
It is a factory of Query, Criteria and Transaction i.e. it provides factory methods to return these instances.

---

8) Is Session a thread-safe object?

No, Session is not a thread-safe object, many threads can access it simultaneously. In other words, you can share it between threads.

9) What is the difference between session.save() and session.persist() method?

No.	save()	persist()
1)	returns the identifier (Serializable) of the instance.	return nothing because its return type is void.
2)	Syn: public Serializable save(Object o)	Syn: public void persist(Object o)

---

10) What is the difference between get and load method?

The differences between get() and load() methods are given below.

No.	get()	load()
1)	Returns **null** if object is not found.	Throws **ObjectNotFoundException** if object is not found.
2)	get() method always **hit the database**.	load() method **doesn't hit** the database.
3)	It returns real object **not proxy**.	It returns **proxy object.**
4)	It should be used if **you are not sure** about the existence of instance.	It should be used if **you are sure** that instance exists.

---

11) What is the difference between update and merge method?

The differences between update() and merge() methods are given below.

No.	update() method	merge() method
1)	Update means to edit something.	Merge means to combine something.
2)	update() should be used if session doesn't contain an already persistent state with same id. It means update should be used inside the session only. After closing the session it will throw error.	merge() should be used if you don't know the state of the session, means you want to make modification at any time.

Let's try to understand the difference by the example given below:

...

```
SessionFactory factory = cfg.buildSessionFactory();
Session session1 = factory.openSession();

Employee e1 = (Employee) session1.get(Employee.class, Integer.valueOf(101));//passing id of employee
session1.close();
```

```
e1.setSalary(70000);

Session session2 = factory.openSession();
Employee e2 = (Employee) session1.get(Employee.class, Integer.valueOf(101));//passing same id

Transaction tx=session2.beginTransaction();
session2.merge(e1);

tx.commit();
session2.close();
```

After closing session1, e1 is in detached state. It will not be in session1 cache. So if you call update() method, it will throw an error.
Then, we opened another session and loaded the same Employee instance. If we call merge in session2, changes of e1 will be merged in e2.

12) What are the states of object in hibernate?

There are 3 states of object (instance) in hibernate.

1. **Transient**: The object is in transient state if it is just created but has no primary key (identifier) and not associated with session.
2. **Persistent**: The object is in persistent state if session is open, and you just saved the instance in the database or retrieved the instance from the database.
3. **Detached**: The object is in detached state if session is closed. After detached state, object comes to persistent state if you call lock() or update() method.

13) What are the inheritance mapping strategies?

There are 3 ways of inheritance mapping in hibernate.

1. Table per hierarchy
2. Table per concrete class
3. Table per subclass

---

14) How to make a immutable class in hibernate?

If you mark a class as mutable="false", class will be treated as an immutable class. By default, it is mutable="true".

---

15) What is automatic dirty checking in hibernate?

The automatic dirty checking feature of hibernate, calls update statement automatically on the objects that are modified in a transaction.
Let's understand it by the example given below:

```
...
SessionFactory factory = cfg.buildSessionFactory();
Session session1 = factory.openSession();
Transaction tx=session2.beginTransaction();
```

```
Employee e1 = (Employee) session1.get(Employee.class, Integer.valueOf(101));

e1.setSalary(70000);

tx.commit();
session1.close();
```

Here, after getting employee instance e1 and we are changing the state of e1.
After changing the state, we are committing the transaction. In such case, state will be updated automatically. This is known as dirty checking in hibernate.

---

16) How many types of association mapping are possible in hibernate?

There can be 4 types of association mapping in hibernate.

1. One to One
2. One to Many
3. Many to One
4. Many to Many

---

17) Is it possible to perform collection mapping with One-to-One and Many-to-One?

No, collection mapping can only be performed with One-to-Many and Many-to-Many

---

18) What is lazy loading in hibernate?

Lazy loading in hibernate improves the performance. It loads the child objects on demand.
Since Hibernate 3, lazy loading is enabled by default, you don't need to do lazy="true". It means not to load the child objects when parent is loaded.

---

19) What is HQL (Hibernate Query Language)?

Hibernate Query Language is known as an object oriented query language. It is like structured query language (SQL).
The main advantage of HQL over SQL is:

1. You don't need to learn SQL
2. Database independent
3. Simple to write query

---

20) What is the difference between first level cache and second level cache?

No.	First Level Cache	Second Level Cache
1)	First Level Cache is **associated with Session**.	Second Level Cache is associated with **SessionFactory**.
2)	It is **enabled** by default.	It is **not enabled** by default.

Hibernate Interview Questions and Answers

1. **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.

2. **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.

3. **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.

4. 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.
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.

5. 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.Connection` and 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.

6. 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.

7. 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.

8. Name some important annotations used for Hibernate mapping?

Hibernate supports JPA annotations and it has some other annotations in`org.hibernate.annotations` package. 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.

```
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
}
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
}
```

9. 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.

10. 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.

11. 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.

12. 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.

13. 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.

```
<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.

14. 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.

15. 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.

16. 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.

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

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-ehcache</artifactId>
    <version>4.3.5.Final</version>
</dependency>
```

- Add below properties in hibernate configuration file.

```
<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>
```

- Create EHCache configuration file, a sample file myehcache.xml would look like below.

```
<?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>

```

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

```
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 for a complete example with explanation.

17. What are different states of an entity bean?

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

0. **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().
1. **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.
2. **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().

18. 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.

19. 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.

20. 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 get`HibernateException`.

21. 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.
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.

22. What are the collection types in Hibernate?

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

0. Bag
1. Set
2. List
3. Array
4. Map

23. How to implement Joins in Hibernate?

There are various ways to implement joins in hibernate.

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

24. 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.

25. 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.

26. 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");
```

27. 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.

28. 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.

29. 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.

30. 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.

31. 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:

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

Learn some quick examples at Hibernate Criteria Example.

32. How to log hibernate generated sql queries in log files?

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

```
<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.

33. 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.

34. 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.

0. Hibernate One to One Mapping
1. Hibernate One to Many Mapping
2. Hibernate Many to Many Mapping

35. 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.

36. 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.

```
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:

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

37. 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.

- Add log4j dependencies for maven project, if not maven then add corresponding jar files.
- 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.
- For standalone projects, use static block to configure log4j using `DOMConfigurator` or`PropertyConfigurator`. 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 and Servlet log4j example.

38. 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.

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

For a complete example, go through Hibernate JNDI DataSource Example.

39. 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.

0. Add hibernate-entitymanager, hibernate-core and spring-orm dependencies.
1. 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.
2. If you are using Hibernate 3, you need to configure`org.springframework.orm.hibernate3.LocalSessionFactoryBean` or`org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean` in Spring Bean configuration file. For Hibernate 4, there is single class`org.springframework.orm.hibernate4.LocalSessionFactoryBean` that should be configured.
3. 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 and Spring MVC Hibernate Integration.

40. 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.

41. 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.

42. Which design patterns are used in Hibernate framework?

Some of the design patterns used in Hibernate Framework are:

- Domain Model Pattern – An object model of the domain that incorporates both behavior and data.
- 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.
- Proxy Pattern for lazy loading
- Factory pattern in SessionFactory

43. What are best practices to follow with Hibernate framework?

Some of the best practices to follow in Hibernate are:

- Always check the primary key field access, if it's generated at the database layer then you should not have a setter for this.
- 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)`.
- 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.
- Use native sql query only when it can't be done using HQL, such as using database specific feature.
- If you have to sort the collection, use ordered list rather than sorting it using Collection API.
- 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.
- For web applications, always try to use JNDI DataSource rather than configuring to create connection in hibernate.
- Avoid Many-to-Many relationships, it can be easily implemented using bidirectional One-to-Many and Many-to-One relationships.
- For collections, try to use Lists, maps and sets. Avoid array because you don't get benefit of lazy loading.
- 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.
- Prefer DAO pattern for exposing the different methods that can be used with entity bean
- Prefer lazy fetching for associations

44. 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 atHibernate Validation Example.

45. What is the benefit of Hibernate Tools Eclipse plugin?

Hibernate Tools plugin helps us in writing hibernate configuration and mapping files easily. The major benefit is the content assist to help us with properties or xml tags to use. It also validates them against the Hibernate DTD files, so we know any mistakes before hand. Learn how to install and use atHibernate Tools Eclipse Plugin.

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

This one is one of the tricky Hibernate Interview questions because Session's lock() method reattach object without synchronizing or updating with the database. So you need to be very careful while using lock() method. By the way, you can always use Session's update() method to sync with the database during reattachment. Sometimes this Hibernate question is also asked as what is difference between Session's lock() and update() method. You can use this key point to answer that question as well. See Java Persistence with Hibernate for more details.

Both of these methods and saveOrUpdate() method are intended for reattaching a detached object.
The session.lock() method simply reattaches the object to the session without checking or updating the database on the assumption that the database in sync with the detached object.
It is the best practice to use either session.update(..) or session.saveOrUpdate().
Use session.lock() only if you are absolutely sure that the detached object is in sync with your detached object or if it does not matter because you will be overwriting all the columns that would have changed later on within the same transaction.

47.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.

48.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.

49. When we are updating a record, we can use session.flush() with Hibernate. What's the need for flush()?

As rightly said in above answers, by calling flush() we force hibernate to execute the SQL commands on Database. But do understand that changes are not "committed" yet. So after doing flush and before doing commit, if you access DB directly (say from SQL prompt)and check the modified rows, you will NOT see the changes.

50. Explain Isolation Level?

I have read about 4 levels of isolation:

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
READ UNCOMMITTED	Permitted	Permitted	Permitted
READ COMMITTED	--	Permitted	Permitted
REPEATABLE READ	--	--	Permitted
SERIALIZABLE	--	--	--

I want to understand the lock each transaction isolation takes on the table
READ UNCOMMITTED - no lock on table
READ COMMITTED - lock on committed data
REPEATABLE READ - lock on block of sql(which is selected by using select query)

SERIALIZABLE - lock on full table(on which Select query is fired)
below are the three phenomena which can occur in transaction isolation Dirty Read- no lock Nonrepeatable Read - no dirty read as lock on committed data Phantom Read - lock on block of sql(which is selected by using select query)
For example, you have 3 concurrent process A, B and C. A starts a transaction, writes data and commit/rollback (depending on results). B just executes a SELECT statement to read data. C reads and updates data. All these process work on same table T.

- READ UNCOMMITTED - no lock on table. You can read data in the table while writing on it. This means, A writes data (uncommited) and B can read this uncommited data and use it (for any purpose). If A executes a rollback, B still has read the data and used it. This is the fastest but most insecure way to work with data since can lead to data holes in not physically related tables (yes, two tables can be logically but not physically related in real world apps =\).
- READ COMMITTED - lock on committed data. You can read the data that was only commited. This means, A writes data and B can't read the data saved by A until A executes a commit. The problem here is that C can update data that was read and used on B and B client won't have the updated data.
- REPEATABLE READ - lock on block of sql(which is selected by using select query). This means, B reads the data under some condition i.e. WHERE aField > 10 AND aField < 20, A inserts data where aField value is between 10 and 20, then B reads the data again and get a different result.
- SERIALIZABLE - lock on full table(on which Select query is fired). This means, B reads the data and **no other transaction can modify the data** on the table. This is the most secure but slowest way to work with data. Also, since a simple read operation locks **the table**, this can lead to heavy problems on production: imagine that T table is an Invoice table, user X wants to know the invoices of the day and user Y wants to create a new invoice, so while X executes the read of the invoices, Y can't add a new invoice (and when it's about money, people get really mad, specially the bosses).

I want to understand where we define these isolation levels : only at jdbc/hibernate level or in DB also
Using JDBC, you define it using Connection#setTransactionIsolation.
Using Hibernate:

```
<property name="hibernate.connection.isolation">2</property>
```

Where

- 1: READ UNCOMMITTED
- 2: READ COMMITTED
- 4: REPEATABLE READ
- 8: SERIALIZABLE