***What is Hibernate?***

🡪*Hibernate is a Java framework that simplifies the development of Java application to interact with the database.*

*🡪Hibernate is ORM(Object Relational Mapping) tool.*

*🡪ORM tools help to connect/map Object oriented programming language with relational databases.*

*🡪It is an Open source, lightweight*

🡪Also it is non-invasive(will not force) framework, means it wont force the programmer to extend/implement any class / interface (Loose Coupling)

🡪*It was introduced by Gavin King in 2001.*

*🡪We can build any type of application using Hibernate Framework.*

***DAO Layer 🡪 Data Access Objects***

*Ctrl+Shift+? 🡪 Add cmments to xml file*

*3 core Interfaces of Hibernate :-*

***1)SessionFactory-*** *Allows us to create instances of Sessions, allows us to create multiple sessions.It is a core Interface of hibernate. It is used to create a connection between java application and database.It acts like a Bridge between application and databases.*

*(FactoryDesignPattern).*

***Connection Properties required:-***

*1)Driver,*

*2) URL,*

*3)Username,*

*4)Password*

*are the properties we require to create Connection.*

***Hibernate Properties required:-***

* ***hibernate.Dialect*** *(We use it to tell hibernate that we should use SQLDialect,PostgreSqlDialect) So that it can decide which type of query hibernate should create.Datatype conversion. Convert Hibernate related data to SQL type of data.*
* ***hibernate.hbm2ddl\_auto*🡪*create/update/:-*** *Used to Automatically convert hibernate to DDL commands.*

***Create****🡪creates new table each and every time the code is executed, at backend it runs:*

*Used for testing purposes*

***Drop table if exists tableName***

***Create table tableName();***

***Update🡪*** *Creates only once and after that whenever the code is executed it just updates/alters the table*

* ***hibernate.show\_sql*🡪*true:-*** *Used to display which query just got executed its not a compulsion to print the query but it’s a good practice.*

***Xml(Extensible markup Language)🡪*** *format to write data, data is written in Tags.*

*For Older Projects 1.0:-*

1. ***Create Xml file***🡪*hibernate.cfg.xml*🡪*Add DTD(Document-Type-Definition)*🡪*org.hibernate.dtd*🡪*Copy dtd 3lines of code of cfg file and paste in our hibernate.cfg.xml file*
2. *Hibernate-configuration tag🡪SessionFactory tag 🡪Connection Property (property tags)🡪Hibernate Property(property tags)🡪 mapping resource tag (mention hbm mapping file name)*
3. ***Create Xml file***🡪*hibernate.hbm.xml*🡪*Add DTD(Document-Type-Definition)*🡪*org.hibernate.dtd*🡪*Copy dtd 3lines of code of hbm file and paste in our hibernate.hbm.xml file*
4. *Hibernate-mapping tag 🡪 mention class name inside class tag(table will be created with that class name🡪Mention primary key with Id Tag🡪add rest of properties with datatypes*

*a)Create configuration object with .configure() Method*

*b)Configuration has a method called buildSessionFactory() which creates instance of SessionFactory and store it in SessionFactory.*

*c)SessionFactory has opensession methodwhich returns an instance of session which needs to be stored in Session object.*

*d)session has save method which saves data to the table*

*e)session has begin transaction and commit method to permanently save the data in database table.*

***\*(We use commit only for DML Operations(insert,update,delete) and commit is not used for DQL commands)***

***2)Session*** *– It is used to create connection between application and database. It provides various methods to communicate with database and application. And perform CRUD opertions*

***\*Auto-commit is False by default in Hibernate***

***\*Annotations provide extra information about variable, method , class to JVM at runtime.***

***Similar to Marker Interface(empty interface) as they also provide more information at runtime to JVM.***

***--------------------------------------------------------------------------------------------------------------------------------------***

***CRUD Operations***

***1)session .save(object) method*** *🡪 It is method used to pass the object from Transient stage to Persistent stage. beginTransaction() is mandatory 🡪* ***Returns Serializable int i.e ID***

***Transient stage->****Only object is created with new keyword and available in Java application only then it is knows as Transient stage.*

***Persistent stage->****When Object with data is present in Java application as well as database, it is known as Persistent stage.*

***2)session.persist(object) 🡪*** *This method used to pass the object from Transient stage to Persistent stage.beginTransaction() is not mandatory 🡪* ***Returns void***

***3) session.get(typeOfObject.class, int id)***

*🡪 This method returns data with help of primary key/ID only.*

*This method returns the object passed as parameter i.e Object of POJO class*

*We need to pass POJO class name for which we want to get the data.*

*It doesn’t sends proxy data it sends actual data and does eager-loading*

*We will use this method when we are not sure about data availability inside database.*

*It will hit the database/schema each and everytime.*

*Reduces application performance, it gets slow down ,because need to fetch and search all data*

***If any object is not present with the specified Id then it will print “ null “***

*Eg:- Student student=session.get(Student.class,3)*

*🡪This will return student object data present on id no 3*

***4) session.load(typeOfObject.class,int id)***

***🡪*** *This method returns data with help of primary key/ID only.*

*This method returns the object passed as parameter i.e Object of POJO class*

*We need to pass POJO class name for which we want to get the data.*

*It sends proxy data and does lazy-loading.*

*We will use this method when we are sure about data availability inside database.*

*It doesn’t fire any query until and unless any fetching/display method is not invoked*

***If any object is not present with the specified Id then it will throw “ ObjectNotFoundException “***

*High application performance, as we know and are familiar with data availability in database.*

***5) session.update(object):-***

***🡪 Returns void***

*Used to make modifications in database*

*It is DML query we need to beginTransaction() and commit() as well.*

*Update query gets fired at backend by hibernate.*

***6) session.saveOrUpdate(object) :-***

*🡪* ***Returns void***

*If ID is of Generated Type then we cannot insert any random ID at any place inside database.*

*First checks if provided object Id is available or not****, if not present*** *it creates/inserts one record saves it with mentioned ID but for this the ID shouldn.t be of Generated Type.*

*It is DML query we need to beginTransaction() and commit() as well.*

***If present*** *then directly updates the existing object which we provided as parameter to method.*

***7) session.delete(object) :-***

*🡪* ***Returns void***

*It is used to delete whole object from database.*

*It is DML query we need to beginTransaction() and commit() as well.*

*--------------------------------------------------------------------------------------------------------------------------------------sf.openSession() 🡪 Each and everytime it is going to create new instance of session.*

*sf.getCurrentSession() 🡪It first checks if any other session is created or opened*

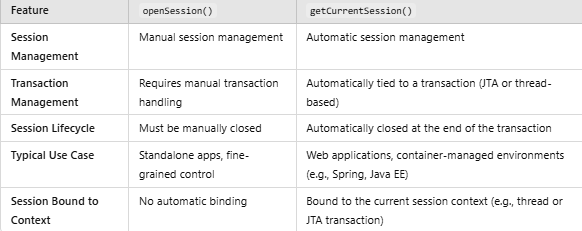
***If Yes****🡪 it uses the same session for next operations and return that instance only.*

***If No****🡪 it creates new session and returns that session only*

***To achieve this functionality:-***

*We need to add one Hibernate property inside HibernateUtils/ Configuration file*

*Environment.CURRENT\_SESSION\_CONTEXT\_CLASS 🡪 “ thread ”*



***I.Q) Why do we require default constructor in Entity class / POJO class?***

*🡪* ***JPA****/Hibernate does not know how to use parameterized constructors to create an object because it does not have access to the data it needs to pass to such constructors.*

*Hibernate uses getInstance() of Instance class which requires Default constructor to create an Object. Even if we provide parameterized constructor in our class, still we need to provide Default constructor in hibernate Entity class.*

*The default constructor allows the framework to instantiate the class and then set the field values (e.g., using setters or reflection) after the object is created.*

***Ways to create Object in JAVA:-***

*1) new keyword*

*2) clone method*

*3) Deserialization*

*4) Factory Design Pattern*

*5) getInstance()*

*6) Reflection API’S.*

***What is Cache and why do we need it?***

* *We need cache because it improves performance of application.*
* *Caching is a phenomena of pooling of an object, where copies of object which share same session are created for future use.*

***First-level Cache / Session Cache:-***

* *It is also called as Session-cache.*
* *The first-level cache is* ***session-bound****, meaning it exists only within the scope of a Hibernate Session. Once the session is closed, the cache is discarded, and all cached entities are lost.*
* *Every time a new session is created, a new first-level cache is created.*
* *When you load an entity using Hibernate, it is stored in the first-level cache. If the same entity is requested again within the same session, Hibernate does not need to query the database again; instead, it retrieves the object from the cache*
* *FirstLevel cache is by default ON in Hibernate*
* *session.clear(): Clears the entire first-level cache, removing all entities and their state from the cache.*
* *session.evict(entity): Evicts (removes) a specific entity from the first-level cache, even though it still exists in the database.*

***Second-level Cache / Session-Factory Cache:-***

* *To use this cache we need to add extra vendor externally i.e we need to add jars of that vendor.*
* *Extra property needs to be added 🡪 optional🡪 ehcache🡪select all jars and add to project*

*Environment.CACHE\_REGION\_FACTORY, ”org.hibernate.cache.ehcache.internal.EhcacheRegionFactory”*

* *Add annotation to that class where we want 2nd level cache*

*🡪@cache(usage=CacheConcurrencyStrategy.READ\_ONLY) 🡪provides read only access*

* *This cache operates at the session factory level and is shared across multiple sessions. It allows Hibernate to store objects that are loaded from the database so that they don't need to be reloaded from the database on subsequent queries, even if the session is closed.*
* *The L2 cache allows data to be shared across sessions, making it especially useful in scenarios where multiple sessions or users access the same data.*
* *Also useful in heavy applications.*

|  |  |
| --- | --- |
| ***1st Level Cache / Session Cache*** | ***2nd Level Cache / SessionFactory Cache*** |
| *Present at Session Level* | *Present at SessionFactory level* |
| *By-default it is ON* | *We need to add externally (jars, annotation)* |
| *No Vendor is needed* | *External vendor needed* ***Eg:- ehcache*** |
| *No annotation is needed* | *@cache(usage=CacheConcurrencyStrategy.READ\_ONLY) is used* |
| *session.clear()-clears entire cache* | *Cache cache=Sf.getCache() – getsAllCache cache.evictAl()l- clear all cache* |
| *Session.evict(entity)-clears particular cache* | *Cache.evict(classname) – class for which we have added @cache annotation that class cache will be cleared* |
| *We cannot stop this cache , switch off this cache* | *We can stop this cache, switch off this cache by removing @cache annotations and all other properties.* |

***Real-life Examples of 1st Level Cache:-***

***1.* *Short-lived Data:***

* ***Use Case****: L1 cache is useful for short-lived or frequently accessed data that does not need to be shared across multiple sessions.*
* ***Example****: A user logging into a social media application and browsing their recent posts or notifications. Once the session ends, the data becomes irrelevant, and the cache is cleared.*
* ***Benefit****: Since L1 is session-based, the cache is automatically cleared when the session ends, ensuring that memory usage is optimized.*

***2. Improving User Experience:***

* ***Use Case****: In interactive applications where a user performs several related operations (e.g., viewing products, adding items to a cart), L1 cache speeds up the access to the same objects within the same session.*
* ***Example****: In a customer support application, once a support ticket is fetched in a session, subsequent operations like viewing or updating the ticket will be fast, without querying the database multiple times.*
* ***Benefit****: Faster response times and a smoother user experience.*

***Real-life Examples of 2nd Level Cache:-***

1. ***Large Applications with Multiple Modules:***

* ***Use Case****: In large applications (like ERP or CRM systems), different modules may require access to the same entities, and the L2 cache can optimize data retrieval across multiple users and modules.*
* ***Example****: A CRM system may have multiple modules (sales, marketing, customer support) that frequently access common customer data (names, contact info). With L2 cache, once customer data is loaded, it can be reused by all modules, reducing redundant database queries.*
* ***Benefit****: Centralized data storage in the cache improves overall performance and reduces redundancy, making the system more scalable and efficient.*

1. ***Data that Changes Infrequently:***

* ***Use Case****: For data that doesn't change often but is accessed frequently, such as configuration settings, product catalog details, or reference data, L2 cache is ideal.*
* ***Example****: In an online retail application, the product catalog (e.g., product types, categories) is relatively static and rarely changes. Once it's loaded into the L2 cache, it can be used by multiple users or across multiple sessions without hitting the database.*
* ***Benefit****: Significant reduction in database queries and improved application performance.*

***Mapping :-***

***One -To-One*** *(Unidirectional)*

*Eg:- 1) Car Has-A Engine 2) Person Has-A Passport 3)Person Has-A AadharCard*

***One -To-Many*** *(Unidirectional)*

*Eg:- 1) Company Has Employee 2) Teacher Has Students*

*Why do we need normalization?*

*🡪To avoid data redundancy and increase data integrity in the table we need normalization.*

*What is dirty checking?*

*🡪Dirty checking is one of the drawback of hibernate, where it doe unwanted checking of columns , that is termed as Dirty checking.*

***Difference between JDBC and Hibernate***

|  |  |
| --- | --- |
| ***JDBC*** | ***Hibernate*** |
| *In JDBC we need to write code manually to map Object’s model class data to schema of relational model.* | *Hibernate maps the object model’s data to schema of database itself with help of annotations* |
| *JDBC enables developers to create queries and update data to a database using the Structured Query Language (SQL)* | *Hibernate uses HQL (Hibernate Query Language) which is similar to SQL but understands object-oriented concepts like inheritance, association etc.* |
| *Code needs to be written in a try-catch block as it throws checked exceptions (SQLexception).* | *Whereas Hibernate manages the exceptions itself by marking them as unchecked.* |
| *Creating associations between relations is quite hard in JDBC.* | *Associations like 1->1, 1->M, M->1, and M->M can be acquired easily with the help of annotations* |
| *It is a database connectivity tool. Means an API* | *It is an Java Framework which is collection of many different API’s* |
| *It has low performance than Hibernate* | *It has high performance* |
| *One needs to maintain explicitly database connections and transactions.* | *It itself manages its own transactions.* |
| *Does not supports cache mechanism* | *Supports cache mechanism It has First-level Caching, Second-level Caching* |
| *Statement, Prepared Statement, Callable Statement is used to build and execute queries.* | *By default supports Prepared Statement.* |
| *Primary Key , Foreign Keys are maintained.* | *Using different annotations we can create and maintain different keys. Annotations like(Auto, Identity, Sequence, Table per class)* |
| *Manually we need to create tables* | *Automatic table creation by using ORM Tool* |
| ***Boilerplate Code:*** *Requires writing a lot of repetitive code for connection handling, query execution, result processing, and exception handling.* | ***Reduces Boilerplate Code:*** *No need to write repetitive JDBC code.* |
| ***Manual Mapping:*** *You have to manually map result sets to Java objects.* | ***Automatic Mapping:*** *Converts Java objects to database tables and vice versa.* |
| ***Tight Coupling****: SQL queries are embedded in Java code, making changes difficult.* | ***Loose Coupling :*** *There are less dependancies , has-A relationship provides loose coupling* |

**What is loose Coupling?**

Loose coupling refers to a design principle in software development where components (classes, modules, or systems) have minimal dependencies on each other. This makes the system more flexible, easier to maintain, and scalable.

**Why is Loose Coupling Important?**

1. **Easier Maintenance:** Changes in one module have minimal impact on others.
2. **Better Reusability:** Components can be reused independently.
3. **Scalability & Flexibility:** Easier to modify or extend the system.
4. **Improved Testability:** Components can be tested separately (unit testing).

**Example: TV & Remote Control**

* ***Tight Coupling:*** *If the remote is designed to work* ***only*** *with a specific TV brand, changing the TV means buying a new remote.*
* ***Loose Coupling:*** *A universal remote works with multiple TV brands because it uses a* ***standardized interface****.*

***We*** *don’t need to write DAO layer classes , instead hibernate will do it for us , we will just tell hibernate that we need to save this object, then Hibernate will automatically save it for us.*

*When we will create a class an that time itself we will map that class to the table , and all fields inside that class will act as columns inside the table.*

*Hibernate is based on Jdbc only at the backend, just the difference is we get ease in accessing and manipulating data without writing complex queries which we do in MySQL.*

***✅ Q1. What is Hibernate and why is it used?***

*🔀* ***Tricky/Indirect Variations****:*

* *Why would you prefer Hibernate over JDBC in enterprise apps?*
* *How do you achieve database abstraction in Java applications?*

*🟩* ***Definition*** *Hibernate is an* ***open-source ORM (Object-Relational Mapping)*** *tool for Java that simplifies the interaction between Java classes and relational databases.*

*🟩* ***Why Hibernate?***

* ***Eliminates boilerplate JDBC code*** *(e.g., connection handling, result set traversal).*
* *Supports* ***automatic schema generation****,* ***lazy loading****, and* ***caching****.*
* *Promotes* ***database independence****—Java code remains untouched even if the DB changes from MySQL to PostgreSQL.*
* *Provides HQL (Hibernate Query Language), which is* ***database-agnostic****.*

*🛠️* ***Real-Time Example*** *In a retail app, let’s say we have a Product class. Without Hibernate, inserting it into the DB needs manual SQL. With Hibernate:*

*java*

*Product p = new Product("MacBook Pro", 120000);*

*session.save(p); // Hibernate auto-generates SQL and maps to DB*

*Hibernate internally generates SQL like:*

*sql*

*INSERT INTO products (name, price) VALUES ('MacBook Pro', 120000);*

***✅ Q2. What are the key advantages of using Hibernate over JDBC?***

*💡 Tricky Alternatives / Indirect Ways to Ask:*

* *Why would you prefer Hibernate instead of writing plain JDBC code?*
* *What benefits does Hibernate offer in a large-scale enterprise application?*
* *How does Hibernate reduce developer workload compared to traditional JDBC?*

*🟩* ***Advantages of Hibernate over JDBC:***

*🔹* ***Eliminates Boilerplate Code*** *Hibernate reduces the amount of code by handling repetitive tasks like establishing connections, managing transactions, and executing SQL queries.*

*🔹* ***Automatic Table Mapping*** *You can directly map Java classes to DB tables using annotations or XML—no need to write SQL INSERT, SELECT, etc.*

*🔹* ***Database Independence*** *Hibernate supports* ***dialects****, allowing seamless switching between databases (MySQL → Oracle) without changing Java code.*

*🔹* ***Caching Mechanism*** *Hibernate provides* ***first-level and second-level caching****, which reduces database hits and improves performance.*

*🔹* ***Lazy Loading*** *Only fetches data when needed—helps optimize performance by loading only required objects.*

*🔹* ***Built-in Transaction Management*** *Hibernate integrates with* ***JTA and JDBC*** *for managing transactions without explicitly writing complex code.*

*🔹* ***HQL Support*** *Hibernate Query Language (HQL) is object-oriented, more readable, and avoids syntax issues of raw SQL.*

*🛠️* ***Real-Time Example:***

*Let’s say you’re building a* ***Banking Application****.*

*Using JDBC:*

*java*

*Connection con = DriverManager.getConnection(...);*

*PreparedStatement ps = con.prepareStatement("INSERT INTO accounts (name, email) VALUES (?, ?)");*

*ps.setString(1, "Rahul");*

*ps.setString(2, "rahul@example.com");*

*ps.executeUpdate();*

*With Hibernate:*

*java*

*Accounts acc = new Accounts();*

*acc.setName("Rahul");*

*acc.setEmail("rahul@example.com");*

*Session session = sessionFactory.openSession();*

*session.beginTransaction();*

*session.save(acc);*

*session.getTransaction().commit();*

*✅ No need to manage connection, query, or result set manually—****Hibernate abstracts it all.***

***✅ Q3. What is the difference between get() and load() in Hibernate?***

*💡 Tricky Alternatives / Indirect Ways to Ask:*

* *When do you prefer get() over load()?*
* *Which method will throw an exception if no record is found?*
* *Can you explain Hibernate's proxy mechanism?*

*🟩* ***Difference Between get() and load():***

| ***Feature*** | ***get()*** | ***load()*** |
| --- | --- | --- |
| *Object Retrieval* | *Immediately hits the database* | *Returns a* ***proxy*** *(lazy fetch)* |
| *If record not found* | *Returns null* | *Throws ObjectNotFoundException* |
| *Use Case* | *When object must exist* | *When lazy loading is fine (not used yet)* |
| *Proxy Use* | *Doesn’t use proxy* | *Returns a proxy object* |

*✨* ***Key Highlights:***

* *get() is eager fetching – hits the DB immediately.*
* *load() is lazy fetching – hits DB only when the object is accessed.*
* *load() is more efficient but risky if you access a non-existent record.*

*🛠️* ***Real-Time Example:***

*java*

*Session session = sessionFactory.openSession();*

*Accounts acc1 = session.get(Accounts.class, 1); // Hits DB immediately*

*Accounts acc2 = session.load(Accounts.class, 2); // Returns proxy*

*System.out.println(acc2.getName()); // Hits DB here for 'load'*

*If ID 2 does not exist, load() will throw an exception at access time.*

***✅ Q4. What are the different states of an object in Hibernate?***

*💡 Tricky Alternatives / Indirect Ways to Ask:*

* *When is an object considered detached in Hibernate?*
* *Can you explain the lifecycle of a persistent object?*
* *What happens when you modify an object outside a session?*

*🟩* ***Object States in Hibernate:***

1. ***Transient*** *– Object is newly created but not associated with a Hibernate session.*
2. ***Persistent*** *– Object is associated with an open session, and any change will be tracked.*
3. ***Detached*** *– Object was persistent, but the session is now closed. Changes won’t be tracked.*

*✨* ***Key Highlights:***

* *Changes in* ***Persistent state*** *are automatically synchronized with the database.*
* *Detached objects need to be* ***reattached*** *using update() or merge().*

*🛠️* ***Real-Time Example:***

*java*

*Accounts acc = new Accounts(); // Transient*

*Session session = sessionFactory.openSession();*

*session.beginTransaction();*

*session.save(acc); // Persistent*

*session.getTransaction().commit();*

*session.close(); // Detached*

*acc.setEmail("newemail@x.com"); // Not tracked anymore*

***✅ Q5. What is Hibernate Caching and its types?***

*💡 Tricky Alternatives / Indirect Ways to Ask:*

* *How does Hibernate improve performance via caching?*
* *What's the difference between first-level and second-level cache?*
* *Can you explain Hibernate's caching strategy?*

*🟩* ***Types of Caching in Hibernate:***

1. ***First-Level Cache (Session Cache):***
   * *Enabled by default.*
   * *Works at the Session level.*
   * *Same object is returned within the session, no DB hit.*
2. ***Second-Level Cache (SessionFactory Cache):***
   * *Optional, needs configuration.*
   * *Shared across sessions using EHCache, Redis, etc.*
3. ***Query Cache:***
   * *Caches query results.*
   * *Needs second-level cache to be enabled.*

*✨* ***Key Highlights:***

* *Caching reduces database calls and improves performance.*
* *Use* ***@Cacheable*** *for enabling entity-level second-level cache.*

*🛠️* ***Real-Time Example:***

*java*

*Session session1 = sessionFactory.openSession();*

*Accounts a1 = session1.get(Accounts.class, 10); // Hits DB*

*Accounts a2 = session1.get(Accounts.class, 10); // No DB hit (1st level cache)*

*session1.close();*

*Session session2 = sessionFactory.openSession();*

*Accounts a3 = session2.get(Accounts.class, 10); // DB hit again (unless 2nd level enabled)*

*If* ***2nd level cache*** *is enabled, a3 would be fetched from cache even in a new session.*

***✅ Q6. What is the difference between save(), persist(), and saveOrUpdate()?***

*💡 Tricky Alternatives / Indirect Ways to Ask:*

* *Which method would you use to avoid inserting duplicate records?*
* *What's the difference between save() and persist() in terms of transaction behavior?*
* *Can you insert or update the same entity with a single method?*

*🟩* ***Difference Between save(), persist(), and saveOrUpdate():***

| ***Method*** | ***Returns*** | ***Transaction Required?*** | ***Inserts Duplicate?*** | ***Updates?*** |
| --- | --- | --- | --- | --- |
| *save()* | *Serializable (ID)* | *Not mandatory* | *Yes (if duplicate)* | *No* |
| *persist()* | *void* | ***Must*** *be in TX* | *Exception on duplicate* | *No* |
| *saveOrUpdate()* | *void* | *Not mandatory* | *No (checks existence)* | *✅ Yes* |

*✨* ***Key Highlights:***

* *save() is flexible but may cause* ***duplicate inserts*** *if used improperly.*
* *persist() is* ***JPA standard*** *and requires active transaction.*
* *saveOrUpdate() is best when you're unsure whether the object exists in DB.*

*🛠️* ***Real-Time Example:***

*java*

*Product p = new Product("Laptop", 80000);*

*// save() returns generated ID*

*Serializable id = session.save(p);*

*// persist() just saves without returning ID*

*session.persist(p);*

*// saveOrUpdate() will insert or update as needed*

*session.saveOrUpdate(p);*

***✅ Q7. What is the role of Configuration, SessionFactory, and Session in Hibernate?***

*💡 Tricky Alternatives / Indirect Ways to Ask:*

* *How does Hibernate initialize and manage DB connections?*
* *Can you explain the object creation hierarchy in Hibernate?*
* *What’s the purpose of SessionFactory?*

*🟩* ***Hibernate Core Objects:***

1. ***Configuration***
   * *Reads hibernate.cfg.xml or annotations.*
   * *Sets up properties like dialect, DB credentials.*
2. ***SessionFactory***
   * *Heavyweight object created* ***once per application****.*
   * *Thread-safe, used to create Session objects.*
3. ***Session***
   * *Lightweight and* ***not thread-safe****.*
   * *Represents a single unit of work with DB.*

*✨* ***Key Highlights:***

* *SessionFactory is* ***expensive to create****, reuse it.*
* *Always open/close Session per request to avoid memory issues.*

*🛠️* ***Real-Time Example:***

*java*

*Configuration cfg = new Configuration().configure();*

*SessionFactory factory = cfg.buildSessionFactory();*

*Session session = factory.openSession();*

*Used in most Hibernate-based apps to bootstrap DB operations.*

***✅ Q8. What are the advantages of Hibernate over JDBC?***

*💡 Tricky Alternatives / Indirect Ways to Ask:*

* *Why do most modern Java projects prefer Hibernate?*
* *What limitations of JDBC are solved by Hibernate?*
* *Is Hibernate better for DB portability?*

*🟩* ***Advantages of Hibernate over JDBC:***

| ***Feature*** | ***Hibernate*** | ***JDBC*** |
| --- | --- | --- |
| *Boilerplate Code* | *Eliminates most SQL boilerplate* | *Requires verbose SQL* |
| *ORM Support* | *Yes (object-table mapping)* | *No* |
| *Database Independence* | *✅ High* | *Low (SQL dialect dependency)* |
| *Caching* | *Built-in* | *Manual* |
| *Auto Schema Generation* | *Yes* | *No* |
| *Transaction Handling* | *Integrated with JTA, Spring, etc.* | *Manual* |

*✨* ***Key Highlights:***

* *Hibernate increases* ***productivity*** *and* ***maintainability****.*
* *You write Java objects, not SQL code.*
* *Supports* ***lazy loading, cascading, and associations*** *easily.*

*🛠️* ***Real-Time Example:***

*With JDBC:*

*java*

*PreparedStatement ps = conn.prepareStatement("INSERT INTO Product VALUES (?, ?)");*

*ps.setString(1, "Phone");*

*ps.setDouble(2, 50000);*

*ps.executeUpdate();*

*With Hibernate:*

*java*

*Product p = new Product("Phone", 50000);*

*session.save(p); // No SQL written!*

***✅ Q9. What is the difference between update() and merge()?***

*🎯 Tricky versions:*

* *Which method throws exception if the entity already exists in the session?*
* *How would you reattach a detached object?*

*🟩* ***Comparison Table****:*

| ***Feature*** | ***update()*** | ***merge()*** |
| --- | --- | --- |
| *Behavior* | *Re-attaches object to session* | *Copies state to new persistent instance* |
| *Exceptions* | *Throws if object already in session* | *Safe with existing instance* |
| *Return Type* | *void* | *Returns managed instance* |

*🛠️* ***Real-Time Example****:*

*java*

*Student s = session.get(Student.class, 10);*

*session.close(); // s is detached*

*Session newSession = factory.openSession();*

*newSession.update(s); // Error if s already in session*

*// Better alternative:*

*Student managed = (Student) newSession.merge(s); // Safe*

*✨ Prefer merge() when you're unsure about session state of the object.*

***✅ Q10. What is lazy loading in Hibernate?***

*🎯 Tricky versions:*

* *What happens if you access a lazily loaded collection after session closes?*
* *How does Hibernate optimize performance?*

*🟩* ***Definition****: Lazy loading means* ***data is not loaded until it is accessed****. Hibernate loads associations like lists/maps only when explicitly called.*

*🛠️* ***Real-Time Example****:*

*java*

*@OneToMany(fetch = FetchType.LAZY)*

*private List<Order> orders;*

*java*

*Customer c = session.get(Customer.class, 1);*

*c.getOrders(); // Triggers DB hit only now*

*✨ Reduces memory usage and improves performance  
🚫 But causes* ***LazyInitializationException*** *if accessed outside session.*

***✅ Q11. What is the difference between get() and load() methods in Hibernate?***

*🎯 Tricky Versions:*

* *How would you lazily fetch an entity by ID?*
* *What happens if the record doesn’t exist in get() vs load()?*
* *Which method throws exception on missing record?*

*📘* ***Core Difference:***

| ***Feature*** | ***get()*** | ***load()*** |
| --- | --- | --- |
| ***Fetching*** | *Immediate* | *Lazy (proxy)* |
| ***Return*** | *Returns null if not found* | *Throws ObjectNotFoundException* |
| ***Use Case*** | *Use when record may not exist* | *Use when record must exist* |

*🛠️* ***Real-Time Example:***

*java*

*Session session = factory.openSession();*

*Employee emp1 = session.get(Employee.class, 1); // Immediate DB hit*

*Employee emp2 = session.load(Employee.class, 2); // Returns proxy*

*⚠️* ***Possible Exception:*** *load() throws ObjectNotFoundException if the object does not exist and you try to access a property.*

***✅ Q12. What is the difference between save() and persist()?***

*🎯 Tricky Versions:*

* *How are transient objects added to the database?*
* *Which method returns the primary key?*
* *What if I want JPA-compliant object persistence?*

*📘* ***Key Differences:***

| ***Feature*** | ***save()*** | ***persist()*** |
| --- | --- | --- |
| ***Return Type*** | *Returns Serializable ID* | *void* |
| ***Standard*** | *Hibernate-specific* | *JPA-compliant* |
| ***Outside Txn*** | *Works outside transaction* | *Throws exception outside transaction* |
| ***Duplicate Calls*** | *Can insert duplicates* | *Respects entity uniqueness* |

*🛠️* ***Real-Time Example:***

*java*

*session.save(emp); // Returns generated ID*

*session.persist(emp); // Doesn’t return ID, but inserts entity*

*⚠️* ***Possible Exception:*** *persist() throws TransactionRequiredException if called outside an active transaction.*

***✅ Q13. What is the difference between merge() and update()?***

*🎯 Tricky Versions:*

* *How do you reattach a detached object?*
* *Which method creates a copy of the object?*
* *How to avoid NonUniqueObjectException when updating?*

*📘* ***Key Differences:***

| ***Feature*** | ***update()*** | ***merge()*** |
| --- | --- | --- |
| ***Behavior*** | *Reattaches existing object* | *Copies data to a managed object* |
| ***Exception Risk*** | *Throws NonUniqueObjectException* | *Safe from multiple objects with same ID* |
| ***Return Value*** | *void* | *Returns managed instance* |

*🛠️* ***Real-Time Example:***

*java*

*Employee detachedEmp = new Employee(1, "Alex");*

*// update*

*session.update(detachedEmp); // Risky if object already in session*

*// merge*

*Employee managedEmp = session.merge(detachedEmp); // Safe*

*⚠️* ***Possible Exception:*** *update() throws org.hibernate.NonUniqueObjectException if an object with the same identifier is already in the session.*

***✅ Q14. What is a proxy object in Hibernate?***

*🎯 Tricky Versions:*

* *What does Hibernate use for lazy loading?*
* *What happens if you access a lazy-loaded object outside session?*
* *How does Hibernate reduce database hits?*

*📘* ***Definition:***

*A* ***proxy object*** *is a subclass created by Hibernate using* ***CGLIB*** *or* ***Javassist****. It is used to implement* ***lazy loading****, where actual DB hit is delayed until you access a property.*

*🛠️* ***Real-Time Example:***

*java*

*Employee emp = session.load(Employee.class, 1);*

*System.out.println(emp.getId()); // No DB hit*

*System.out.println(emp.getName()); // DB hit occurs here*

*⚠️* ***Possible Exception:*** *Accessing a lazy proxy* ***after session is closed*** *leads to LazyInitializationException.*

***✅ Q15. What is Hibernate caching? What are its types?***

*🎯 Tricky Versions:*

* *How does Hibernate reduce redundant DB calls?*
* *What's the role of EhCache or Redis in Hibernate?*
* *Difference between session-level and app-level cache?*

*📘* ***Definition:***

*Caching in Hibernate improves performance by reducing the number of database hits for frequently accessed data.*

*🟩* ***Types of Caching:***

1. ***First-Level Cache (L1):***
   * *Enabled by default*
   * *Session-scoped*
   * *Same object loaded twice in session → fetched only once*
2. ***Second-Level Cache (L2):***
   * *Optional, configured separately*
   * *SessionFactory-scoped*
   * *Shared across sessions*
3. ***Query Cache:***
   * *Caches query results*
   * *Must be enabled explicitly*

*🛠️* ***Real-Time Example:***

*java*

*session.get(Employee.class, 1); // First DB hit*

*session.get(Employee.class, 1); // Fetched from L1 cache*

*sessionFactory.getCache().evictAll(); // Clears L2 cache*

*⚠️* ***Possible Exception:*** *Improper configuration of 2nd level cache may result in* ***inconsistent stale data****.*

***✅ Q16. What is the difference between First-Level Cache and Second-Level Cache in Hibernate?***

*🎯 Tricky Versions:*

* *What's the difference between session-scoped and SessionFactory-scoped caching?*
* *Why is the same object not fetched twice in a session?*
* *How does Hibernate manage cache across sessions?*

*📘* ***Key Differences:***

| ***Feature*** | ***First-Level Cache (L1)*** | ***Second-Level Cache (L2)*** |
| --- | --- | --- |
| ***Scope*** | *Per Session* | *Per SessionFactory (shared)* |
| ***Enabled By Default*** | *Yes* | *No (needs config + provider like EhCache)* |
| ***Customizable*** | *No* | *Yes – supports eviction, expiry, etc.* |
| ***Storage*** | *In-memory* | *Pluggable (EhCache, Infinispan, etc.)* |

*🛠️* ***Real-Time Example:***

*java*

*// First-level cache example*

*Employee e1 = session.get(Employee.class, 1); // Hits DB*

*Employee e2 = session.get(Employee.class, 1); // Fetched from L1 cache (no DB hit)*

*// Second-level cache (after enabling it)*

*Session session2 = factory.openSession();*

*Employee e3 = session2.get(Employee.class, 1); // Fetched from L2 cache*

*⚠️* ***Possible Issues:***

* *Stale data if L2 cache is not synchronized.*
* *Needs extra configuration; incorrect setup may cause caching errors.*

***✅ Q17. What is Lazy vs Eager loading in Hibernate?***

*🎯 Tricky Versions:*

* *How does Hibernate fetch associated collections?*
* *When does Hibernate hit the DB for related entities?*
* *What’s the risk of Lazy loading outside a session?*

*📘* ***Definition:***

* ***Lazy Loading****: Fetch data* ***only when accessed****. Saves performance.*
* ***Eager Loading****: Fetch data* ***immediately****, along with the parent.*

*🟩* ***Example:***

*java*

*@OneToMany(fetch = FetchType.LAZY)*

*private List<Order> orders;*

*@OneToMany(fetch = FetchType.EAGER)*

*private List<Order> orders;*

*🛠️* ***Real-Time Scenario:***

*java*

*Employee e = session.get(Employee.class, 1);*

*e.getOrders(); // Triggers DB hit only if LAZY*

*⚠️* ***Possible Exception:***

* *LazyInitializationException if you access a LAZY collection* ***after session is closed****.*

***✅ Q18. What are the states of an object in Hibernate?***

*🎯 Tricky Versions:*

* *What is the lifecycle of an entity?*
* *How does Hibernate manage entity state transitions?*
* *When does Hibernate sync the object with DB?*

*📘* ***Object States:***

1. ***Transient****: Not associated with session or DB.*
2. ***Persistent****: Bound to session; any change is auto-tracked.*
3. ***Detached****: Once persistent, but now disconnected from session.*
4. ***Removed****: Marked for deletion.*

*🛠️* ***Real-Time Example:***

*java*

*Employee e = new Employee(); // Transient*

*session.save(e); // Persistent*

*session.evict(e); // Detached*

*session.delete(e); // Removed*

*⚠️* ***Possible Exception:***

* *LazyInitializationException if you access data in detached state.*
* *TransientObjectException if saving an unsaved associated object.*

***✅ Q19. What is the difference between Evict() and Clear() methods?***

*🎯 Tricky Versions:*

* *How to remove object from Hibernate cache?*
* *Difference between clearing specific vs entire session cache?*
* *What if I want to detach a single object?*

*📘* ***Key Differences:***

| ***Method*** | ***Description*** |
| --- | --- |
| *evict(obj)* | *Removes a* ***single object*** *from session cache* |
| *clear()* | *Clears* ***entire session cache*** |

*🛠️* ***Real-Time Example:***

*java*

*Employee emp = session.get(Employee.class, 1);*

*session.evict(emp); // Only emp is detached*

*session.clear(); // All managed objects are detached*

*⚠️* ***Caution:*** *After evict() or clear(), the object becomes* ***detached****, and further changes won’t be tracked.*

***✅ Q20. What is HQL? How is it different from SQL?***

*🎯 Tricky Versions:*

* *How do you query using entity names instead of table names?*
* *What if I want DB-independent querying in Hibernate?*
* *Can you write object-oriented queries in Hibernate?*

*📘* ***Definition:***

***HQL (Hibernate Query Language)*** *is an* ***object-oriented*** *query language similar to SQL but operates on* ***entity objects****, not tables.*

*🟩* ***Differences with SQL:***

| ***Feature*** | ***HQL*** | ***SQL*** |
| --- | --- | --- |
| *Works On* | *Entity/Property names* | *Tables/Columns* |
| *DB Independent* | *Yes* | *No* |
| *Return Type* | *Objects* | *ResultSet/Rows* |

*🛠️* ***Real-Time Example:***

*java*

*// HQL*

*Query q = session.createQuery("FROM Employee WHERE name = :n");*

*q.setParameter("n", "John");*

*List<Employee> list = q.list();*

*⚠️* ***Possible Exception:***

* *QuerySyntaxException if wrong entity or property is used.*
* *NonUniqueResultException when multiple results are returned for uniqueResult().*

*✅* ***Q21. What is the difference between get() and load() methods in Hibernate?***

*🎯 Tricky Versions:*

* *What’s the difference in fetching objects using get() and load() in Hibernate?*
* *How does Hibernate behave when fetching a non-existing record?*

*🟩* ***Definition:***

* *get()* ***immediately hits the database*** *and returns the object or null if not found.*
* *load() returns a* ***proxy object (lazy loading)*** *and throws an exception if the record doesn’t exist.*

*✨* ***Key Differences:***

* *get() → Eager loading, returns null if not found*
* *load() → Lazy loading, returns a proxy, throws ObjectNotFoundException if accessed and not found*

*🛠️* ***Real-Time Example:***

*java*

*Session session = factory.openSession();*

*// Using get() → returns null if ID doesn't exist*

*Employee emp1 = session.get(Employee.class, 999);*

*System.out.println(emp1); // null if ID 999 doesn't exist*

*// Using load() → returns proxy, throws exception if ID doesn't exist when accessed*

*Employee emp2 = session.load(Employee.class, 999);*

*System.out.println(emp2.getName()); // ⚠️ Throws ObjectNotFoundException*

*⚠️* ***Possible Exception:***

* *load() may throw org.hibernate.ObjectNotFoundException when accessing non-existing records.*

*✅* ***Q22. What is lazy loading in Hibernate?***

*🎯 Tricky Versions:*

* *How does Hibernate delay loading of relationships?*
* *When are associated entities loaded on demand?*

*🟩* ***Definition:*** *Lazy loading is a technique where* ***associated data is not loaded from the database until it is actually accessed****.*

*✨* ***Why it’s used:***

* *Saves memory and boosts performance*
* *Reduces unnecessary joins and fetches*

*🛠️* ***Real-Time Example:***

*java*

*@Entity*

*class Employee {*

*@OneToMany(fetch = FetchType.LAZY)*

*private List<Address> addresses;*

*}*

*Accessing employee.getAddresses() won’t load the data* ***until the call is made****, saving DB hits on object creation.*

*⚠️* ***Possible Exception:***

* *If session is closed before accessing the lazily loaded field, Hibernate throws:  
  LazyInitializationException*

*✅* ***Q23. What is eager loading in Hibernate?***

*🎯 Tricky Versions:*

* *When does Hibernate immediately load associations?*
* *What loading type is opposite to lazy loading?*

*🟩* ***Definition:*** *Eager loading means that* ***associated data is fetched immediately*** *along with the primary entity.*

*✨* ***Use it when****: Associated data is always required and frequent DB hits are inefficient.*

*🛠️* ***Real-Time Example:***

*java*

*@Entity*

*class Employee {*

*@OneToMany(fetch = FetchType.EAGER)*

*private List<Address> addresses;*

*}*

*Here, addresses will be fetched* ***along with Employee*** *when the session queries for an Employee.*

*⚠️* ***Possible Concern:***

* *May lead to* ***performance issues*** *due to unnecessary joins or over-fetching.*

*✅* ***Q24. What is the difference between first-level and second-level cache in Hibernate?***

*🎯 Tricky Versions:*

* *How does Hibernate cache entities per session vs across sessions?*
* *What types of caching does Hibernate support?*

*🟩* ***Definition:***

* ***First-level cache:*** *Default, exists per session.*
* ***Second-level cache:*** *Optional, exists across multiple sessions, backed by providers like Ehcache, Infinispan.*

*✨* ***Key Points:***

* *First-level = always enabled*
* *Second-level = needs configuration*

*🛠️* ***Real-Time Example:***

*java*

*Session session1 = factory.openSession();*

*Employee emp1 = session1.get(Employee.class, 1); // Hit DB*

*Employee emp2 = session1.get(Employee.class, 1); // Fetched from first-level cache*

*Session session2 = factory.openSession();*

*Employee emp3 = session2.get(Employee.class, 1); // DB hit again unless second-level cache is enabled*

*⚠️* ***Note:***

* *If not properly configured,* ***second-level cache won't work****.*
* *Use @Cache(usage = CacheConcurrencyStrategy.READ\_ONLY) for cache-enabled entities.*

*✅* ***Q25. What is the N+1 select problem in Hibernate?***

*🎯 Tricky Versions:*

* *Why do multiple select queries get fired for associations?*
* *What happens if lazy loading is poorly managed?*

*🟩* ***Definition:*** *The N+1 select problem happens when:*

* *1 query fetches the main entity (1 query)*
* *N additional queries are fired for associated entities (N queries)*

*✨* ***Problem Example:***

*java*

*List<Employee> empList = session.createQuery("from Employee").list();*

*for(Employee emp : empList) {*

*System.out.println(emp.getAddress().getCity()); // fires one query per address*

*}*

*🛠️* ***Real-Time Solution:*** *Use JOIN FETCH to avoid it:*

*java*

*String hql = "SELECT e FROM Employee e JOIN FETCH e.address";*

*⚠️* ***Performance Tip:*** *Always analyze SQL logs to detect N+1 issues and refactor with JOIN FETCH or batch fetching.*

*✅* ***Q26. What is the difference between save() and persist() methods in Hibernate?***

*🎯 Tricky Versions:*

* *Which method should you prefer when saving an entity — save() or persist()?*
* *What’s the behavioral difference in terms of return value and transaction scope?*

*🟩* ***Definition:***

* *save() is from Hibernate and returns the* ***generated identifier*** *(Serializable).*
* *persist() is from JPA and* ***does not return anything****. It works within* ***transaction boundaries****.*

*✨* ***Key Differences:***

| ***Aspect*** | ***save()*** | ***persist()*** |
| --- | --- | --- |
| *API origin* | *Hibernate-specific* | *JPA standard* |
| *Return value* | *Returns Serializable ID* | *void* |
| *Use outside txn* | *Allowed* | *Throws Exception* |
| *Persistence timing* | *Immediate or deferred* | *Only within a transaction* |

*🛠️* ***Real-Time Example:***

*java*

*// save()*

*Serializable id = session.save(new Product("Laptop", 45000));*

*System.out.println("ID: " + id); // Prints generated ID*

*// persist()*

*session.persist(new Product("Phone", 25000)); // No return value*

*⚠️* ***Possible Exception:***

* *persist() throws TransactionRequiredException if used* ***outside of an active transaction****.*

*✅* ***Q27. What is the difference between merge() and update() in Hibernate?***

*🎯 Tricky Versions:*

* *How do you handle detached objects in Hibernate?*
* *When should merge() be preferred over update()?*

*🟩* ***Definition:***

* *update() reattaches a* ***detached entity*** *and syncs changes to the DB.*
* *merge() copies changes from a detached entity to a* ***new persistent object*** *and returns it.*

*✨* ***Key Differences:***

| ***Feature*** | ***update()*** | ***merge()*** |
| --- | --- | --- |
| *Entity state* | *Re-attaches detached entity* | *Copies state to managed instance* |
| *Exception risk* | *NonUniqueObjectException* | *Safe from such exception* |
| *Return value* | *void* | *Returns the updated managed object* |

*🛠️* ***Real-Time Example:***

*java*

*Product detachedProduct = new Product(1, "Tablet", 20000);*

*// update()*

*session.update(detachedProduct); // Must not already exist in session*

*// merge()*

*Product mergedProduct = session.merge(detachedProduct);*

*⚠️* ***Possible Exception:***

* *update() may throw org.hibernate.NonUniqueObjectException if the same entity exists in session.*

*✅* ***Q28. What is the role of the SessionFactory in Hibernate?***

*🎯 Tricky Versions:*

* *What is SessionFactory and why is it important?*
* *How is Hibernate's SessionFactory different from JDBC Connection?*

*🟩* ***Definition:*** *SessionFactory is a* ***thread-safe, heavyweight object*** *in Hibernate that creates Session objects.*

*✨* ***Responsibilities:***

* *Reads configuration and mappings*
* *Caches generated SQL statements*
* *Manages database connections efficiently*

*🛠️* ***Real-Time Example:***

*java*

*SessionFactory factory = new Configuration().configure().buildSessionFactory();*

*Session session = factory.openSession();*

*✨* ***Key Point:***

* *It should be created* ***only once per application*** *(usually at app startup).*

*⚠️* ***Exception Tip:***

* *Misconfigured XML or annotations will throw MappingException or ConfigurationException during build.*

*✅* ***Q29. What is HQL and how is it different from SQL?***

*🎯 Tricky Versions:*

* *What language does Hibernate use to query objects?*
* *How does HQL provide abstraction over SQL?*

*🟩* ***Definition:*** *HQL (Hibernate Query Language) is an* ***object-oriented query language*** *similar to SQL but operates on* ***Java classes*** *and not table names.*

*✨* ***Key Differences from SQL:***

* *Works with entity/class names instead of table names*
* *Automatically supports joins using mapped associations*
* *Portable across databases*

*🛠️* ***Real-Time Example:***

*java*

*// HQL*

*String hql = "FROM Product p WHERE p.price > 10000";*

*List<Product> products = session.createQuery(hql, Product.class).list();*

*⚠️* ***Note:***

* *Syntax errors in HQL throw QuerySyntaxException.*

*✅* ***Q30. What are Named Queries in Hibernate?***

*🎯 Tricky Versions:*

* *How can you define reusable queries in Hibernate?*
* *What is the purpose of @NamedQuery?*

*🟩* ***Definition:*** *Named Queries are* ***static HQL or native SQL queries*** *defined with annotations or XML and reused by name across the application.*

*✨* ***Benefits:***

* ***Performance optimization*** *via pre-compilation*
* *Promotes DRY code (Don’t Repeat Yourself)*
* *Simplifies large-scale applications*

*🛠️* ***Real-Time Example:***

*java*

*@Entity*

*@NamedQuery(name = "Product.byCategory", query = "FROM Product p WHERE p.category = :category")*

*public class Product { ... }*

*// Usage*

*Query<Product> query = session.createNamedQuery("Product.byCategory", Product.class);*

*query.setParameter("category", "Electronics");*

*⚠️* ***Possible Mistake:***

* *If query is not defined or named incorrectly, you get IllegalArgumentException: Named query not known.*

*✅* ***Q31. What are Transient, Persistent, and Detached states in Hibernate?***

*🎯 Tricky Versions:*

* *Can you explain the lifecycle of a Hibernate entity?*
* *What are different object states in the Hibernate context?*

*🟩* ***Definition:***

*Hibernate defines 3 main object states during the lifecycle of an entity:*

| ***State*** | ***Meaning*** |
| --- | --- |
| ***Transient*** | *Not associated with Hibernate session or database* |
| ***Persistent*** | *Associated with an active Hibernate session and database* |
| ***Detached*** | *Was persistent once, but now disconnected from session* |

*🛠️* ***Real-Time Example:***

*java*

*Product product = new Product("TV", 30000); // Transient*

*Session session = factory.openSession();*

*session.beginTransaction();*

*session.save(product); // Becomes Persistent*

*session.getTransaction().commit();*

*session.close(); // Now Detached*

*⚠️* ***Possible Mistake:***

* *Operating on a detached object without merging or updating it may result in* ***stale data or data loss****.*

*✅* ***Q32. What is Lazy Loading in Hibernate?***

*🎯 Tricky Versions:*

* *How can you optimize Hibernate performance for child collections?*
* *What does Hibernate fetch lazily?*

*🟩* ***Definition:*** *Lazy loading is a technique in Hibernate where associated data (collections or entities) is* ***loaded only when accessed****, not when the parent is loaded.*

*✨* ***Benefit:*** *Improves performance by reducing unnecessary database queries.*

*🛠️* ***Real-Time Example:***

*java*

*@Entity*

*public class Product {*

*@OneToMany(mappedBy = "product", fetch = FetchType.LAZY)*

*private List<Review> reviews;*

*}*

*⚠️* ***Common Exception:***

* *LazyInitializationException: When a lazily loaded entity is accessed* ***outside*** *the session scope.*

*✅* ***Q33. What is Eager Loading in Hibernate?***

*🎯 Tricky Versions:*

* *How can you preload all associated data in one go?*
* *What’s the opposite of Lazy Fetching?*

*🟩* ***Definition:*** *Eager loading means that* ***all related entities/collections are fetched immediately*** *when the parent is fetched.*

*🛠️* ***Real-Time Example:***

*java*

*@Entity*

*public class Product {*

*@OneToMany(mappedBy = "product", fetch = FetchType.EAGER)*

*private List<Review> reviews;*

*}*

*✨* ***When to Use:*** *Use EAGER when associated data is always required, such as* ***master-detail reports*** *or* ***dashboards****.*

*⚠️* ***Drawback:***

* ***Performance overhead****: May lead to unnecessary data loading and complex joins.*

*✅* ***Q34. What are the differences between get() and load() methods in Hibernate?***

*🎯 Tricky Versions:*

* *Which method throws an exception if entity doesn’t exist?*
* *What’s the behavioral difference between get() and load()?*

*🟩* ***Definition:***

| ***Method*** | ***Behavior*** |
| --- | --- |
| *get()* | *Returns null if no record is found* |
| *load()* | *Returns a proxy object and throws ObjectNotFoundException if not found* |

*🛠️* ***Real-Time Example:***

*java*

*// get()*

*Product p1 = session.get(Product.class, 1); // Null if not found*

*// load()*

*Product p2 = session.load(Product.class, 1); // Throws exception if not found*

*✨* ***When to Use:***

* *Use get() when unsure if entity exists.*
* *Use load() for performance and when* ***existence is guaranteed****.*

*✅* ***Q35. What is a Hibernate Proxy Object?***

*🎯 Tricky Versions:*

* *What does Hibernate return before actual DB call?*
* *How does Hibernate implement Lazy Initialization internally?*

*🟩* ***Definition:*** *A* ***Proxy Object*** *is a subclass created by Hibernate for* ***lazy loading****. It* ***defers DB access*** *until a property is accessed.*

*🛠️* ***Real-Time Example:***

*java*

*Product p = session.load(Product.class, 1); // Not loaded from DB yet*

*String name = p.getName(); // Now triggers DB call*

*✨* ***Why Proxies?*** *Improves performance and memory usage for large applications.*

*⚠️* ***Exception Alert:***

* *Accessing proxy after session is closed leads to LazyInitializationException.*

*✅* ***Q36. What is the difference between save(), persist(), and saveOrUpdate() methods in Hibernate?***

*🎯 Tricky Versions:*

* *How does Hibernate behave differently when saving transient entities?*
* *Which method returns the generated identifier?*

*🟩* ***Definition:***

| ***Method*** | ***Behavior*** | ***Returns ID?*** |
| --- | --- | --- |
| *save()* | *Inserts a new record, returns identifier* | *✅ Yes* |
| *persist()* | *Similar to save, but follows JPA — doesn’t return ID* | *❌ No* |
| *saveOrUpdate()* | *Saves if transient, updates if persistent or detached* | *✅ Conditional* |

*🛠️* ***Real-Time Example:***

*java*

*Product p = new Product("Laptop", 50000);*

*// save()*

*Serializable id = session.save(p); // ID returned*

*// persist()*

*session.persist(p); // No ID returned*

*// saveOrUpdate()*

*session.saveOrUpdate(p); // Smart insert/update*

*⚠️* ***Exception Possibilities:***

* *persist() throws EntityExistsException if entity already exists.*
* *save() can result in duplicate inserts if used incorrectly with detached objects.*

*✅* ***Q37. What is the difference between merge() and update()?***

*🎯 Tricky Versions:*

* *Which method avoids exceptions with detached objects?*
* *How do you synchronize a detached object back to the database?*

*🟩* ***Definition:***

| ***Method*** | ***Use Case*** | ***Safe for Detached?*** |
| --- | --- | --- |
| *update()* | *Reassociates a detached object to session* | *❌ No (throws exception if already attached)* |
| *merge()* | *Copies state of detached object into a persistent one* | *✅ Yes* |

*🛠️* ***Real-Time Example:***

*java*

*Product p = new Product(1, "TV", 20000);*

*session.update(p); // May throw exception if already associated*

*session.merge(p); // Safe*

*⚠️* ***Exception:***

* *NonUniqueObjectException with update() if the same object is already in session.*

*✅* ***Q38. What is Hibernate Caching and what are its types?***

*🎯 Tricky Versions:*

* *How does Hibernate reduce DB hits?*
* *What caching strategies does Hibernate support?*

*🟩* ***Definition:*** *Hibernate provides caching to avoid repetitive DB calls for the same data.*

*🟩* ***Types of Caching:***

1. ***First-Level Cache (Session level)*** *– enabled by default.*
2. ***Second-Level Cache (SessionFactory level)*** *– requires configuration.*
3. ***Query Cache*** *– stores result sets of HQL/SQL queries.*

*🛠️* ***Real-Time Example:***

*java*

*Session s1 = sessionFactory.openSession();*

*Product p1 = s1.get(Product.class, 1); // DB hit*

*Product p2 = s1.get(Product.class, 1); // Cached*

*✨* ***Benefits:***

* *Reduces latency*
* *Improves performance*

*⚠️* ***Risk:***

* *Data inconsistency if cache is stale and DB is updated by other sources.*

*✅* ***Q39. How do you enable Second-Level Cache in Hibernate?***

*🎯 Tricky Versions:*

* *How do you cache data beyond session scope in Hibernate?*
* *How do you retain entity across sessions?*

*🟩* ***Steps to Enable:***

1. *Add a cache provider (e.g.,* ***EhCache****,* ***OSCache****,* ***Infinispan****)*
2. *Configure hibernate.cfg.xml:*

*xml*

*<property name="hibernate.cache.use\_second\_level\_cache">true</property>*

*<property name="hibernate.cache.region.factory\_class">*

*org.hibernate.cache.ehcache.EhCacheRegionFactory*

*</property>*

1. *Annotate Entity:*

*java*

*@Cacheable*

*@Cache(usage = CacheConcurrencyStrategy.READ\_ONLY)*

*🛠️* ***Real-Time Example:***

*java*

*@Cacheable*

*@Cache(usage = CacheConcurrencyStrategy.READ\_ONLY)*

*public class Product { ... }*

*✅* ***Q40. What is Query Cache in Hibernate and how is it enabled?***

*🎯 Tricky Versions:*

* *How do you cache the result of HQL queries in Hibernate?*
* *What's the role of query cache?*

*🟩* ***Definition:*** *Query Cache stores the result sets of queries (not entities), reducing DB load on frequently run queries.*

*🟩* ***How to Enable:***

*xml*

*<property name="hibernate.cache.use\_query\_cache">true</property>*

*🛠️* ***Real-Time Example:***

*java*

*List<Product> products = session.createQuery("from Product")*

*.setCacheable(true)*

*.list();*

*✨* ***When to Use:***

* *For frequently used read-only queries like dropdowns, dashboard reports.*

*⚠️* ***Important Note:***

* *Depends on* ***second-level cache*** *to cache entities returned by queries.*

*✅* ***Q41. What is Lazy Loading in Hibernate?***

*🎯 Tricky Versions:*

* *How can Hibernate optimize performance by delaying database calls?*
* *What happens if you access a collection outside of the session?*

*🟩* ***Definition:******Lazy Loading*** *means fetching the associated data* ***only when it's actually needed****, not when the parent entity is loaded.*

*🟩* ***How it Works:*** *Hibernate uses proxy objects to delay the loading of associated collections or objects until a method is called.*

*🛠️* ***Real-Time Example:***

*java*

*@Entity*

*public class Product {*

*@OneToMany(fetch = FetchType.LAZY)*

*private List<Review> reviews;*

*}*

*// Usage*

*Product p = session.get(Product.class, 1); // Reviews not loaded*

*p.getReviews().size(); // Now Hibernate fires the SQL to fetch reviews*

*⚠️* ***Common Exception:***

* *LazyInitializationException: Occurs if you try to access the lazy-loaded data* ***after session is closed****.*

*✅* ***Q42. What is Eager Loading in Hibernate?***

*🎯 Tricky Versions:*

* *When does Hibernate fetch associated entities immediately?*
* *What loading strategy could impact performance?*

*🟩* ***Definition:******Eager Loading*** *fetches the associated data* ***immediately*** *with the main entity, regardless of whether it's needed or not.*

*🛠️* ***Real-Time Example:***

*java*

*@Entity*

*public class Product {*

*@OneToMany(fetch = FetchType.EAGER)*

*private List<Review> reviews;*

*}*

*When a Product is fetched, its reviews will be loaded at the same time.*

*✨* ***Use Case:*** *When you know that associated data will always be needed.*

*⚠️* ***Drawback:*** *Can result in* ***performance issues*** *due to unnecessary joins and data fetching.*

*✅* ***Q43. What is N+1 SELECT problem in Hibernate?***

*🎯 Tricky Versions:*

* *Why does Hibernate generate too many queries with collections?*
* *How can a single fetch lead to multiple queries?*

*🟩* ***Definition:*** *When you fetch a list of entities and then fetch a collection or associated data for each, Hibernate ends up executing* ***1 query for parent + N queries for children****.*

*🛠️* ***Real-Time Example:***

*java*

*List<Product> products = session.createQuery("from Product").list();*

*for(Product p : products) {*

*System.out.println(p.getReviews().size()); // Triggers new query each time!*

*}*

*✨* ***Fix:*** *Use JOIN FETCH to solve it.*

*java*

*List<Product> products = session.createQuery("from Product p JOIN FETCH p.reviews").list();*

*✅* ***Q44. What is the difference between get() and load() in Hibernate?***

*🎯 Tricky Versions:*

* *How does Hibernate retrieve data by primary key?*
* *What’s the impact of using a proxy?*

*🟩* ***Comparison Table:***

| ***Method*** | ***Behavior*** | ***Returns Null if not found?*** | ***Uses Proxy?*** |
| --- | --- | --- | --- |
| *get()* | *Immediately hits DB and returns real object* | *✅ Yes* | *❌ No* |
| *load()* | *Returns a proxy, hits DB on access* | *❌ Throws Exception* | *✅ Yes* |

*🛠️* ***Real-Time Example:***

*java*

*Product p1 = session.get(Product.class, 1); // Immediate fetch*

*Product p2 = session.load(Product.class, 1); // Proxy returned*

*⚠️* ***Exception:***

* *load() throws ObjectNotFoundException if entity doesn’t exist.*

*✅* ***Q45. What is the role of the Configuration class in Hibernate?***

*🎯 Tricky Versions:*

* *How does Hibernate read and build the setup?*
* *Which class is used for bootstrap?*

*🟩* ***Definition:*** *Configuration class is responsible for reading hibernate.cfg.xml, loading mappings, and building SessionFactory.*

*🛠️* ***Real-Time Example:***

*java*

*Configuration cfg = new Configuration();*

*cfg.configure("hibernate.cfg.xml");*

*SessionFactory factory = cfg.buildSessionFactory();*

*✨* ***Internally:***

* *Loads DB connection settings*
* *Maps annotated entity classes*
* *Builds connection pool setup*

*✅* ***Q46. What is a Hibernate Proxy?***

*🎯 Tricky Versions:*

* *How does Hibernate avoid fetching full objects?*
* *What object does Hibernate return with lazy loading?*

*🟩* ***Definition:*** *A* ***proxy*** *is a subclass dynamically generated by Hibernate for lazy loading. It delays actual database fetch until a method is accessed.*

*🛠️* ***Real-Time Example:***

*java*

*Product p = session.load(Product.class, 1); // Returns proxy, not real object yet*

*✨* ***Benefit:*** *Boosts performance by avoiding unnecessary DB access.*

*⚠️* ***Exception:***

* *LazyInitializationException if proxy is accessed after session is closed.*

*✅* ***Q47. What is Cascade in Hibernate? What are its types?***

*🎯 Tricky Versions:*

* *How does Hibernate handle operations on child entities automatically?*
* *What are cascading effects in ORM?*

*🟩* ***Definition:******Cascade*** *allows operations performed on a parent entity (e.g., save, delete) to automatically apply to associated child entities.*

*🟩* ***Cascade Types:***

* *ALL*
* *PERSIST*
* *MERGE*
* *REMOVE*
* *REFRESH*
* *DETACH*

*🛠️* ***Real-Time Example:***

*java*

*@OneToMany(cascade = CascadeType.ALL)*

*private List<Review> reviews;*

*✨ Now if you save a Product, its reviews will be saved too.*

*✅* ***Q48. What is Hibernate Validator?***

*🎯 Tricky Versions:*

* *How does Hibernate ensure bean-level validation?*
* *How to restrict field values using annotations?*

*🟩* ***Definition:*** *Hibernate Validator is the reference implementation of* ***Bean Validation (JSR 380)*** *for validating constraints on object fields.*

*🛠️* ***Real-Time Example:***

*java*

*public class Product {*

*@NotNull*

*private String name;*

*@Min(1)*

*private double price;*

*}*

*🛠️* ***Validation Code:***

*java*

*ValidatorFactory vf = Validation.buildDefaultValidatorFactory();*

*Validator validator = vf.getValidator();*

*Set<ConstraintViolation<Product>> violations = validator.validate(product);*

*✅* ***Q49. How do you integrate Hibernate with Spring?***

*🎯 Tricky Versions:*

* *How is SessionFactory injected in Spring?*
* *How does Spring manage transactions with Hibernate?*

*🟩* ***Integration Steps:***

1. *Add dependencies (spring-orm, hibernate-core)*
2. *Configure SessionFactory in Spring config*
3. *Use @Transactional for transaction management*
4. *Use HibernateTemplate or SessionFactory.getCurrentSession()*

*🛠️* ***Spring XML Config:***

*xml*

*<bean id="sessionFactory" class="LocalSessionFactoryBean">*

*<property name="dataSource" ref="dataSource"/>*

*<property name="hibernateProperties">...</property>*

*</bean>*

*✅* ***Q50. What is the purpose of HibernateUtil class?***

*🎯 Tricky Versions:*

* *How do you make SessionFactory reusable?*
* *What is the best way to encapsulate Hibernate configuration?*

*🟩* ***Definition:*** *A* ***HibernateUtil*** *class is a helper utility to provide a single SessionFactory instance application-wide (singleton).*

*🛠️* ***Real-Time Example:***

*java*

*public class HibernateUtil {*

*private static SessionFactory factory;*

*static {*

*Configuration cfg = new Configuration().configure();*

*factory = cfg.buildSessionFactory();*

*}*

*public static SessionFactory getSessionFactory() {*

*return factory;*

*}*

*}*

*✨* ***Benefit:*** *Reduces boilerplate and centralizes Hibernate config.*