Primary key must-haves:

1. Unique
2. Constant
3. Required

Use @Transient to indicate that a property should not be persisted, also we can used in field will not be stored in the database.

Property Access gets and sets object values through getter /setter methods (put annotations on the getter method)
Field Access gets and sets object values directly from / to the fields (put @Id and @GeneratedValue on the instance variable)

Different states of an entity in the persistence context?
1- Transient Entity: has NO db entity.
2- Managed Entity: managed by persistence context and has db entity.
3- Detached Entity: not managed by persistence context and has db entity.
4- Removed Entity: corresponding db entity is already removed.

Advantage and disadvantages of **Single-Table per Hierarchy** inheritance strategy in JPA?
- Simple, Easy to implement
- Good performance on all queries, polymorphic and non-polymorphic
- Nullable columns / de-normalized schema
- Table may have to contain lots of columns.

Advantage and disadvantages of **Joined Tables** inheritance strategy in JPA?
- Normalized Schema
- Database view is similar to domain view.
- Inserting or updating an entity results in multiple insert or update statements.
- Necessary joins can give lower query performance.

Polymorphism is the ability of a subtype to appear and behave like its supertype.
A polymorphic query is a query for all objects in a hierarchy, independent of their subtype.

@Embeddable
public class Name { private String firstname; private String lastname; }
wrong: it requires hashCode and equals methods to act as a composite key.

@ElementCollection vs **@OneToMany? its** interchangeably?
@ElementCollection maps non-entities (embeddable or primitives) while **@OneToMany** is used to map entities., and no we cannot use it **interchangeably**.

In "Optimistic Concurrency" does the first commit win or the last commit? What happens when a conflict happens?
First commit wins instead of last commit wins which normally happened by the Read Committed level. An StaleObjectStateException is thrown if a conflict would occur.

What is the role of @Version annotation?
@Version annotation I used to specify version column which:
- Should have no semantic value in the table.
- Should be updated by all applications using the db.
Also, can be a timestamp column which:
- may already exists in the class.
- considered less secure.
- may fit the business logic better.
----
3. Given regular JPA mappings (no extra attributes, all default behavior) what performance issue do you see in the following code?
List salesReps = session.createQuery("from SalesRep").list();
for (SalesRep s : salesReps) { Set customers = s.getCustomers();
for (Customer c : customers) { // do something with the customer } }
it produces The N + 1 Problem, a Hibernate executes many selects, it caused by inappropriate lazy loading of data.
In the given example it retrieves salesreps and then work with related customers, So, we have a query to gets the salesreps, then executes another query for each customer related to salesrep (N queries). Total N + 1.
----
4. What is the default for @ManyToOne and @OneToOne associations? Eager or lazy? How about @OneToMany?
The JPA specifies that both @ManyToOne and @OneToOne default to eager loading, and we can set it to fetch lazy by using attribute (fetch = FetchType.LAZY).
While @OneToMany default to lazy.
----
5. Eager loading is always achieved using joins. True or false?
false, to achieve join we have to use join fetch strategy.
----
1. What is the sequence of events when ApplicationContext loads?
- Reads the Spring configuration from xml or class-path scanning or java config.
- Instantiates objects declared in the Spring configuration.
+ Wires objects together with dependency injection
    o Instantiate the object /Call the constructor.
    o Do the injection calling the setter method(s) if declared in config.
    o Call init-method if declared.
- Creates proxy objects when needed
- Call destroy-method when close if declared.
----
2. Prototype beans are lazy by default. TRUE
----
3. Singleton beans are lazy by default. FASLE (Singleton are Eager by default)
----

4. Is this autowired by type?
@Autowired @Qualifier("emailService") private EmailService emailService;
False (it is by Name because @Qualifier)
----
1. In the context of AOP what is a "Pointcut"?
Pointcut is a set of one or more JoinPoint where some advice should be executed, and pointcut expression language is a way of describing that in code.
----
2. In the context of AOP what is an "Advice"?
Advice is the implementation of the crosscutting concern, or we can say It is the action taken by an aspect at a particular join-point.
----
3. In the context of AOP what is "Aspect"?
Aspect is a module which has a set of APIs providing cross-cutting requirements, It determines What crosscutting concern do I execute (=advice) at which locations in the code (=pointcut).
----
4. What are two advantages of AOP?
- No code tangling (Clean separation of business logic and plumbing code)
- No code scattering (Same thing, just from a different perspective)
----
5. What are two disadvantages of AOP?
- No clear overview of which code runs when.
- Make mistakes easily.
- No compile time checking of the pointcut expression because it is in a string format.
- Common issues with proxy-based AOP.s
----
1. What are the 4 different issues we discussed in class with "Session per Operation" anti-pattern?
Session per-operation problems:
- Session Cache is never used.
- Retrieved objects are immediately detached.
+ No way to automatically load related entities.
    - Need to separately load entities – similar to JDBC Code.
    - Not doing so just creates Lazy initialization exceptions.
+ Because a new session is opened and closed each time, a new transaction is also opened and closed:
- Transaction only spans a single operation.
- Transaction never spans a unit of work.
- Essentially creating non-transactional behavior.
----
2. Why does SessionFactory need to be a singleton bean?
SessionFactory should be a singleton bean because the Session Factory is expensive to create.
- Doing so essentially ‘starts’ Hibernate.
- We generally want to start it only once.

- We need a session factory whenever we use Hibernate (to make Sessions), It needs to be available everywhere.

----

3. Does sessionFactory.getCurrentSession() return a session that is unique to the local thread? Think twice before you answer. **TRUE**

----

4. When you commit a transaction inside a ThreadLocal Hibernate session, it leaves the session open. **FALSE**

----

5. With an "Open Session in View" filter, hibernate session is opened before Controller layer is called and stays open until after view is rendered. **TRUE**

----

6. In a Spring/Hibernate Java web application, hibernate runs inside Spring and Spring runs inside the Java web application container. **TRUE**

----

1. Where would you use a Global Transaction?
Global Transactions (Also called XA transactions) are transactions that span multiple transactional resources Such as databases or message busses, which is more common in enterprise applications.

----

2. Explain the "Unrepeatable Read" issue and mention what level of transaction isolation solves that.
It may happen when using **“Read Committed”** isolation level, if a row is read twice during a transaction, the second read might give different data because of a concurrent update,
And **“Repeatable read”** solves this issue which uses both read and write locks for the duration of the tx.

----

3. Explain the "Phantom Read" issue and mention what level of transaction isolation fixes that.
A **Phantom Read** may happen when using “**Repeatable read**” If the same select is executed twice during a transaction the second result set might include more rows than the first due to concurrent inserts, and we can solve it by using “**Serializable**” isolation level which may cause slowness.

----

4. Explain the "Read Committed" transaction isolation level. What does it guarantee?
It is Default setting for most databases, allows multiple transactions to access the same data, but hides non-committed data from other transactions.
- can read data that is newly committed during the transaction.
- it is guaranteed to read only committed data.

----

5. In what scenario is it acceptable to use the "Read Uncommitted" isolation level?
it acceptable to use the "Read Uncommitted" if you know that you will not use a rollback, can we use for single user in desktop application.

----

1. If Spring is not a transaction manager, what does it do with transactions?
Spring is not a transaction manager, need a transaction manager

- JDBC transaction manager
- Hibernate transaction manager
- XA transaction manger (JTA)

Spring provides an abstraction for transaction management
- You declare how the transactions should be managed
- Spring works with the underlying transaction manager of the transactional resource

----

2. Explain how AOP relates to Spring transactions:

We can use AOP to perform some actions and concerns (advise) before, after or around the main method.
this is the same idea as the spring transaction which will begin the transaction at the beginning of the method and commit it at the end. We can do the same using advise around like below

```
@arround()
transaction.begin
call.processed
transaction.commit
```

----

3. What is the difference between transaction propagation NEVER and NOT_SUPPORTED?

NEVER: If the calling method m1() runs in a transaction T1, an exception is thrown if it will call method m2()
NOT_SUPPORTED: If the calling method m1() runs in a transaction T1, then method m2() does not run within this transaction and it will be suspended.

----

4. Explain why propagation REQUIRES_NEW is better for service level methods than REQUIRED

@Transactional(propagation=Propagation.REQUIRED) – If there is not an existing transaction it will create a new transaction. In case if there is already an existing transaction it will not create a new transaction.
@Transactional(propagation=Propagation.REQUIRES_NEW) – Even if there is already an existing transaction it will create a new transaction i.e it will always create a new transaction.

So Service level methods should have their own transaction. So they can be committed without waiting T1 to be committed. In addition, if T1 is rollback for any reason, then T2 was commit successfully it their own transaction.

----

5. Explain the "Unrepeatable Read" issue in TX Isolation.

This issue caused by Read Committed Level of Transaction Isolation Levels occurs when a transaction is unable to read the same object value more than once, the second read might give different data because of a concurrent update.
