**1. What is JPA?** JPA (Java Persistence API) is a Java EE and Java SE specification that describes a management system for saving java objects to relational database tables in a convenient form.

**2. What is the difference between JPA and Hibernate?** Hibernate is one of the most popular open source implementations. JPA only describes rules and APIs, and Hibernate implements these descriptions, however Hibernate has additional features not described in JPA.

**3. Is it possible to use JPA with noSQl databases?** In general, the JPA specification says only about mapping java objects into relational database tables, but there are a number of implementations of this standard for noSql databases: DataNucleus, ObjectDB, and a number of others.

**4. What is the difference between JPA and JDO?** JPA (Java Persistence API) and Java Data Objects (JDO) are two specifications for storing java objects in databases. If JPA is concentrated only on relational databases, then JDO is a more general specification that describes the ORM for any possible bases and repositories. In principle, JPA can be viewed as part of the JDO specification specialized in relational databases, even though the API of these two specifications does not completely match.

**5. What is Entity?** Entity is a lightweight persistent domain object. The main program entity is the entity class, which can also use additional classes that can be used as auxiliary classes or to maintain state of the entity.

**6. Can the Entity class inherit from non-Entity classes?** Can.

**7. Can an Entity class inherit from other Entity classes?** The same can.

**8. Can a non-Entity class inherit from an Entity class?** And this is also permissible.

**9. Can Entity be an abstract class?** Perhaps, at the same time, it retains all the properties of the Entity, except that it cannot be directly initialized.

**10. What JPA requirements for Entity classes can you list (at least six requirements)?**

* Entity class must be annotated with Entity or described in the XML configuration file JPA,
* Entity class must contain a public or protected constructor with no arguments (it can also have constructors with arguments),
* Entity class must be a top-level class (top -level class),
* Entity class cannot be enum or interface,
* Entity class cannot be final class,
* Entity class cannot contain final fields or methods if they participate in mapping
* If an Entity class object is passed by value as a separate object (detached object)
* The Entity class fields should be directly accessible only to the methods of the Entity class and should not be directly accessible to other classes using this entity. Such classes should refer only to methods (getter / setter methods or other business logic methods in the Entity class),
* The Entity class must contain a primary key, that is, an attribute or group of attributes that uniquely defines the record of this Entity class in the database.

**11. What are the two types of elements in Entity classes. Or in other words, list two types of access (access) to the elements of the Entity classes.** JPA indicates that it can work both with properties of classes (property), designed in the style of JavaBeans, or with fields (field), that is, class variables (instance variables). Accordingly, the type of access will be either property access or field access.

**12. What is the attribute of the Entity class in JPA terminology?** JPA indicates that it can work both with properties of classes (property), designed in the style of JavaBeans, or with fields (field), that is, class variables (instance variables). Both types of elements of the Entity class are called attributes of the Entity class.

**13. What data types are allowed in the attributes of the Entity class (fields or properties)?**

* primitive types and their Java wrappers,
* strings,
* any Java serializable types (implementing the Serializable interface),
* enums;
* entity types;
* embeddable classes
* and collection types 1-6

**14. What data types can be used in the attributes included in the primary key of the Entity class (composite or simple) so that the resulting primary key can be used for any database? And in the case of auto-generated primary key (generated primary keys)?**

* primitive types and their Java wrappers,
* strings,
* BigDecimal and BigInteger,
* java.util.Date and java.sql.Date
* In the case of an auto-generated primary key (generated primary keys), only numeric types are allowed.
* If other types of data are used in the primary key, it can work only for some databases, i.e. becomes not portable (not portable).

**Question 15. What is the Embeddable class?** An embeddable class is a class that is not used by itself, only as part of one or more Entity classes. An entity class can contain both single embedded classes and collections of such classes. Also such classes can be used as keys or map values. At run time, each embedded class belongs to only one object of the Entity class and cannot be used to transfer data between the objects of the Entity classes . In general, such a class serves to make the definition of common attributes for several Entity, we can assume that JPA simply embeds into the Entity instead of an object of this class, the attributes it contains.

**16. Can the Embeddable class contain another Embeddable class?** Yes maybe.

**17. Can the Embeddable class contain relationships with other Entity or Entity collections? If it can, are there any restrictions on such relationships?** Maybe, but only if such a class is not used as the primary key or the map key.

**18. What requirements does JPA set for Embeddable classes?**

* Such classes must satisfy the same rules as the Entity classes, except that they do not have to contain a primary key and be marked with the Entity annotation (see question 10),
* The Embeddable class must be marked with the Embeddable annotation or described in the XML configuration file JPA.

**19. What types of connections (relationship) between Entity do you know (list eight types, or specify four types of connections, each of which can be further divided into two types)?**

* **OneToOne** (one-to-one connection, that is, one Entity object can be associated with no more than one object of another Entity),
* **OneToMany** (one-to-many connection, one Entity object can be associated with a whole collection of Entity)
* **ManyToOne** (many to one link, feedback for OneToMany),
* **ManyToMany** (many to many link) Each of which can be divided into two types:
* **Bidirectional**
* **Unidirectional** link to link is established only on one side, that is, in the case of OneToOne AB only Entity A will have link to Entity B, Entity B will not have link to A .

**20. What is Mapped Superclass?** Mapped Superclass is a class from which Entity is inherited, it may contain JPA annotations, however such a class is not Entity, it does not have to fulfill all the requirements set for Entity (for example, it may not contain a primary key). Such a class cannot be used in EntityManager or Query operations. Such a class should be marked with the MappedSuperclass annotation or, respectively, described in the xml file.

**21. What are the three types of Inheritance Mapping Strategies described in JPA?**

**1) one table for the entire inheritance hierarchy ( a single table per class hierarchy ) –** all entity one table, to identify the type of entity is determined by a special column **“discriminator column”** .

**2) unifying strategy ( joined subclass strategy ) –** in this strategy, each entity class stores data in its own table, but only unique columns (not inherited from ancestor classes) and the primary key, and all inherited columns are written to ancestor class tables a relationship is established between these tables.

**3) one table for each class ( table per concrete class strategy ) –** everything is simple here every individual class of successor has its own table

**22. What are the two types of fetch strategies in JPA do you know?**

**1) LAZY –** these fields will be loaded only during the first access to this field,

**2) EAGER –** these fields will be loaded immediately.

**23. What is EntityManager and what are its main functions you can list?** EntityManager is an interface that describes an API for all basic operations on Enitity, data retrieval and other JPA entities. Essentially the main API for working with JPA. Basic operations:

**1) For operations on Entity:** persist (adding Entity under JPA control), merge (updating), remove (delete), refresh (updating data), detach (removing from management JPA), lock (blocking Enity from changes in other thread),

**2) data Preparation:** find (search and retrieval entity), createQuery, createNamedQuery, createNativeQuery , contains, createNamedStoredProcedureQuery, createStoredProcedureQuery

**3) Preparation of other entities JPA:** getTransaction, getEntityManagerFactory, getCriteriaBuilder, getMetamodel, getDelegate

**4) Work with EntityGraph:** createEntityGraph, getEntityGraph

**5) General operations on EntityManager or all Entities:** close, isOpen, getProperties, setProperty, clear.

**24. What are the four lifecycle status of an Entity Instance’s Life Cycle you can list?**

**1) new –** the object has been created but has not yet generated primary keys and has not yet been saved in the database,

**2) managed –** the object has been created, managed by JPA, has generated primary keys,

**3) detached –** the object has been created, but not managed (or no longer managed) JPA,

**4) removed –** the object is created, managed by JPA, but will be deleted after commit’a transaction.

**25. How does the operation persist on Entity objects of each of the four statuses?**

**1)** If the status is Entity new, then it changes to managed and the object will be saved to the database when a transaction is committed or as a result of flush operations,

**2)** If the status is already managed, the operation is ignored, but dependent Entity can change the status to managed, if there are annotations of cascading changes,

**3)** If the status is removed, then it changes to managed,

**4)** If the status is detached, the exception will be thrown right away or at the commit stage of the transaction.

**26. How does the remove operation affect the Entity objects of each of the four statuses?**

**1)** If the status is Entity new, the operation is ignored, however dependent Entity can change the status to removed, if they have cascading change annotations and they had the status managed,

**2)** If the status is managed, then the status changes to removed and the object is recorded in the database removed during commit commit (remove operations will also occur for all cascade-dependent objects),

**3)** If status is removed, then the operation is ignored,

**4)** If detached, exception is thrown right away or at the commit stage of a transaction.

**27. How does the operation merge affect the Entity objects of each of the four statuses?**

**1)** If the detached state, then either the data will be copied to the existing managed entity with the same primary key, or a new managed is created into which the data is copied,

**1)** If the Entity is new, then a new managed entity will be created into which the past data will be copied object,

**2)** If the status is managed, the operation is ignored, but the operation merge will work on the cascade-dependent Entity, if their status is not managed,

**3)** If the status is removed, exception will be thrown immediately or at the commit stage of the transaction.

**28. How does the refresh operation affect the Entity objects of each of the four statuses?**

**1)** If the status is Entity managed, then as a result of the operation all changes from the database of this Entity will be restored, also a refresh of all cascade-dependent objects will occur,

**2)** If the status is new, removed or detached, exception will be thrown out.

**29. How does the detach operation affect the Entity objects of each of the four statuses?**

**1)** If the status is Entity managed or removed, then as a result of the operation, the status of Entity (and all cascade-dependent objects) will become detached.

**2)** If the status is new or detached, then the operation is ignored.

**30. What is the Basic annotation for?** Basic – indicates the simplest type of data mapping on a database table column. Also in the annotation parameters you can specify fetch field access strategy and whether this field is required or not.

**31. What is the Access annotation for?** It defines the access type for an entity class, superclass, embeddable, or individual attributes, that is, how JPA will refer to entity attributes, like class fields (FIELD) or class properties (PROPERTY) that have getters (getter) and setters.

**32. What annotations can overlap connections (override entity relationship) or attributes inherited from the superclass, or specified in the embeddable class when using this embeddable class in one of the entity classes and do not overlap in the others?**

1. AttributeOverride to overlap fields, properties and primary keys,
2. AttributeOverrides can similarly override fields, properties and primary keys with multiple values,
3. AssociationOverride to override entity relationship,
4. AssociationOverrides to overlap multiple relationships.

**33. What annotation can I manage JPA caching for this Entity?**

Cacheable – allows you to enable or disable the use of a second-level cache (second-level cache) for this Entity (if the JPA provider supports caching and cache settings (second-level cache) are ENABLE\_SELECTIVE or DISABLE\_SELECTIVE, see question 41). Note that the property is inherited and if it is not blocked by the heirs, then caching will change for them too.

**34. What annotations are used to set the class for converting the basic attribute of the Entity to another type when saving / retrieving data of their base (for example, work with the attribute of the Entity boolean type, but save it as a number to the base)?Convert and Converts –** allow you to specify a class for converting the Basic attribute of an Entity into another type (Converts – allow you to specify several conversion classes). Classes for conversion must implement the AttributeConverter interface and can be marked (but not required) by the Converter annotation.

**35. What annotation can be used to define a class, the methods of which should be executed under certain JPA operations on an Entity or Mapped Superclass data (such as deleting, changing data, etc.)?**The EntityListeners annotation allows you to set a Listener class that will contain event handling methods (callback methods) defined by the Entity or Mapped Superclass.

**36. What are callback methods in JPA for? To which entities do callback method annotations apply? List seven callback methods (or, equivalently, annotations of callback methods)** Callback methods are used to call on certain Entity events (that is, add processing, for example, deletion of Entity methods by JPA), can be added to the entity class, to the mapped superclass, or to the callback listener class specified by the EntityListeners annotation There are seven callback methods (and annotations with the same names):

**1) PrePersist**

**2) PostPersist**

**3) PreRemove**

**4) PostRemove**

**5) PreUpdate**

**6) PostUpdate**

**7) PostLoad**

**37. What annotations are used to establish the procedure for issuing elements of the Entity collections?**

The OrderBy and OrderColumn annotations are used for this.

**38. What annotation can exclude fields and properties of Entity from mapping (property or field is not persistent)?** For this is the abstract Transient.

**39. What are the six types of locks (lock) described in the JPA specification (or what values ​​does the enum LockModeType in JPA have)?** JPA has six types of locks, list them in order of increasing reliability

**1) NONE –** without blocking

**2) OPTIMISTIC (or READ synonym for JPA 1) –** optimistic blocking

**3) OPTIMISTIC\_FORCE\_INCREMENT (or WRITE synonym, remaining from JPA 1) –** optimistic lock with forced increase of the version field,

**4) PESSIMISTIC\_READ –** pessimistic lock for reading,

**5) PESSIMISTIC\_WRITE –** pessimistic lock for write (and read),

**6) PESSIMISTICI-PII-IZIETIHI, IHIETI , I, I, I, I, I, I, I, I; on record (and the with forced increase in the field of versioning.**

**40. What are the two types of cache (cache) you know in JPA and what are they for?**

**1) first-level cache (first-level cache) —**caches data from a single transaction;

**2) second-level cache (second-level cache) —**caches data for more than one transaction. The JPA provider can, but is not required to implement work with the second-level cache. This kind of cache can save access time and improve performance, but the downside is the ability to get outdated data.

**41. What are the options for setting the second-level cache (second-level cache) in JPA or what similarly describe what values ​​can the shared-cache-mode element take from persistence.xml?**JPA speaks of the five shared-cache-mode values ​​from persistence.xml, which defines how the second-level cache will be used:

**1) ALL –** all Entity can be cached in the second-level cache ,

**2) NONE –** caching is disabled for all Entity,

**3) ENABLE\_SELECTIVE —** caching only works for those Entities that have the Cacheable (true) or their xml equivalent set, for all others, caching is disabled,

**4) DISABLE\_SELECTIVE —** caching works for all Entity, except for those that have the Cacheable annotation (false) or its xml equivalent is

**5) UNSPECIFIED –** caching is not defined, each provider EPA uses its default value for caching,

**42. How can you change the fetch strategy settings of any Entity attributes for individual queries (query) or search methods (find), then if Entity has an attribute with fetchType = LAZY, but for a specific query you need to make it EAGER or vice versa?**For this, the EntityGraph API exists, it is used like this: using the NamedEntityGraph annotation for an Entity, it creates named EntityGraph objects that contain a list of attributes that need to change fetchType to EAGER, and then this name is specified in hits queries or the find method. As a result, the fetchType attribute of the Entity changes, but only for this request. There are two standard properties for specifying EntityGraph in hit:

**1) javax.persistence.fetchgraph –** all attributes listed in EntityGraph change fetchType to EAGER, all others to LAZY

**2) javax.persistence.loadgraph –** all attributes listed in EntityGraph change fetchType to EAGER, all others retain their fetchType (that is, if the attribute not specified in EntityGraph has fetchType was EAGER, then it will remain EAGER) With NamedSubgraph you can also change fetchType nested Entity objects.

**43. How is it possible in the code to work with the second-level cache (delete all or certain Entity from the cache, find out if this Entity has been cached, etc.)?**To work with the second level cache (second level cache) in JPA, the Cache interface is described which contains a large number of methods for managing the second level cache (second level cache), if supported by the JPA provider, of course. The object of this interface can be retrieved using the getCache method from EntityManagerFactory.

**44. How can I get JPA metadata (information about Entity types, Embeddable and Managed classes, etc.)?**JPA uses the Metamodel interface to get this information. The object of this interface can be obtained by the getMetamodel method from an EntityManagerFactory or EntityManager.

**45. What is JPQL (Java Persistence query language) and how is it different from SQL?**JPQL (Java Persistence query language) is a query language, almost the same as SQL, but instead of the names and columns of database tables, it uses the names of the Entity classes and their attributes. The query parameters also use the data types of the attributes of the Entity, and not the database fields.

**46. ​​What is meant by polymorphism (polymorphism) in JPQL queries (Java Persistence query language) and how to “turn it off”?** JPQL queries have automatic polymorphism, that is, each Entity request returns not only the objects of this Entity, but also objects of all its descendant classes, regardless of the inheritance strategy.

**47. What is the Criteria API and what is it used for?** The Criteria API is also a query language, similar to JPQL (Java Persistence query language), but the queries are based on methods and objects, that is, the queries look like this:

|  |  |
| --- | --- |
| 1  2  3  4 | CriteriaBuilder cb = …  CriteriaQuery&lt;Customer&gt; q = cb.createQuery(Customer.class);  Root&lt;Customer&gt; customer = q.from(Customer.class);  q.select(customer); |

**48. What is the difference in Hibernate Entity requirements from the Entity requirements in the JPA specification**

1) A constructor with no arguments in Hibernate is not required to be public or protected, it is recommended that it be at least a package of visibility, but this is only a recommendation, if the Java security settings allow access to private fields, then it can be private.

2) JPA categorically requires not to use final classes, Hibernate only recommends not using such classes so that it can create a proxy for lazy loading, but it allows you to either turn off the Proxy proxy (lazy = false), or use an interface containing all mapping methods of this class (proxy annotation (proxyClass = interface.class))

**49. What is the unique inheritance strategy in Hibernate, but not in the JPA specification?** Unlike JPA, Hibernate has a unique inheritance strategy **called implicit polymorphism.**

**50. What are the main new features in the JPA 2.1 specification compared with JPA 2.?**

**1) Entity Graphs –** a dynamic change fetchType mechanism for each request,

**2) Converters –** a mechanism for defining converters for specifying functions for converting Entity attributes into database fields,

**3) DDL generation –** automatic generation of tables, indexes and schemas,

**4) Stored Procedures –** a mechanism for calling stored procedures from JPA,

**5) Criteria Update / Delete –** a mechanism for invoking bulk updates or deletes using the Criteria API,

**6) Unsynchronized persistence contexts –** an opportunity to specify SynchronizationType,

**7) New features in JPQL / Criteria API :** arithmetic subqueries, generic database functions, join ON clause, TREAT function,