Realtests.1z0-898.63Questions

Number: 1z0-898
Passing Score: 800
Time Limit: 120 min
File Version: 8.2

1z0-898

Java EE 6 Java Persistence API Developer Certified Expert Exam

1. Still valid. Passed with 98%. Questions are word for word. Go through the practice test about 4 times READING the question and understanding the answer will help you a lot.
2. ALL the credit goes to this Excellent and wonderful vce file. Thanks
3. Some new questions thrown in. Some of the other ?s are asked differently and have different answers. Just study all the explanations and you will be fine.
4. This dump most valid.. Take this and prepare..
5. This dump is reliable as well as reasonable.

Exam A

QUESTION 1
A developer has created a deep entity class hierarchy with many polymorphic relationships between entitles. Which inheritance strategy, as defined by the inheritanceType enumerated type, will be most performed in this scenario?

A. Single table-per-class-hierarchy (InheritanceType.SINGLE_TABLE)
B. Joined-subclass (inheritanceType. JOINED)
C. Table-per-concrete-class (inheritanceType.TABLE_PER_CLASS)
D. Polymorphic join table (inheritanceType. POLYMORPHIC_JOIN_TABLE)

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
The Single Table per Class Hierarchy Strategy
This strategy provides good support for polymorphic relationships between entities and queries that cover the entire entity class hierarchy.

Note:Enum InheritanceType:
*SINGLE_TABLE
A single table per class hierarchy.
*JOINED
A strategy in which fields that are specific to a subclass are mapped to a separate table than the fields that are common to the parent class, and a join is performed to instantiate the subclass.

*TABLE_PER_CLASS
A table per concrete entity class.

Incorrect:
Not D: There is no such thing as inheritanceType.POLYMORPHIC_JOIN_TABLE.

References:

QUESTION 2
A developer is creating an entity which is mapped to a table that has a primary key constraint defined on two character columns and would like to use mapping defaults as much as possible to simplify the code.

Which two mapping options can be chosen? (Choose two.)

A. Use an @id property that constructs a private field as a concatenation of two columns.
B. Use a separate class to map those two columns and use an @idclass annotation to denote I primary key field or property in the entity.
C. Use a separate @Embeddable class to map those two columns and use an @EmbeddedId annotation to denote a single primary key field or property in the entity.
D. Use a separate @Embeddable class to map those two column and add two fields or properties the entity, each marked as @id, that correspond to the fields or properties in the embeddable class.
E. Use a separate class to map those two columns. Specify that class using @Idclass annotation on the entity class. Add two fields or properties to the entity, each marked as @Id, that correspond to the fields or properties in that separate class.

Correct Answer: CE
Section: (none)
Explanation

Explanation/Reference:
Explanation:
C:Annotation Type EmbeddedId
Applied to a persistent field or property of an entity class or mapped superclass to denote a composite primary key that is an embeddable class. The embeddable class must be annotated as Embeddable.

E:Annotation Type IdClass
Specifies a composite primary key class that is mapped to multiple fields or properties of the entity.The names of the fields or properties in the primary key class and the primary key fields or properties of the entity must correspond and their types must be the same.

Example:

```
@IdClass(com.acme.EmployeePK.class)
@Entity
public class Employee {
@Id String empName;
@Id Date birthDay;

}
```

References:

QUESTION 3
A developer wants to model the grades for a student as a Map<course, integer>. Assume that Student and Course are entitles, and that grades are modeled by integers.

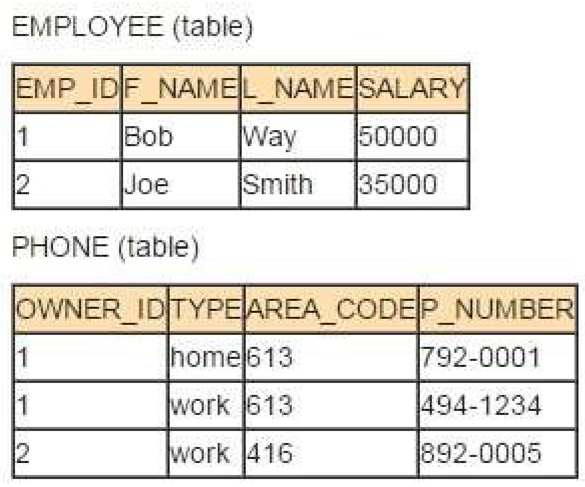
Which of the following two statements are correct? (Choose two)

A. The developer can model the grades as an element collection in the Student entity.
B. The developer can model the grades as a oneToMany relationship in the Student entity.
C. The mapping for the key of the map can be specified by the @MapKeycolumn annotation.
D. The mapping for the value of the map can be specified by the @Column annotation.

Correct Answer: AD
Section: (none)
Explanation

Explanation/Reference:
Explanation:
A:JPA 2.0 defines an ElementCollection mapping. It is meant to handle several non-standard relationship mappings. An ElementCollection can be used to define a one-to-many relationship to an Embeddable object, or a Basic value (such as a collection of Strings). An ElementCollection can also be used in combination with a Map to define relationships where the key can be any type of object, and the value is an Embeddable object or a Basic value.

D:
Example of an ElementCollection relationship database:



EMPLOYEE (table)

EMP_ID	F_NAME	L_NAME	SALARY
1	Bob	Way	50000
2	Joe	Smith	35000

PHONE (table)

OWNER_ID	TYPE	AREA_CODE	P_NUMBER
1	home	613	792-0001
1	work	613	494-1234
2	work	416	892-0005

Example of an ElementCollection relationship annotations @Entity

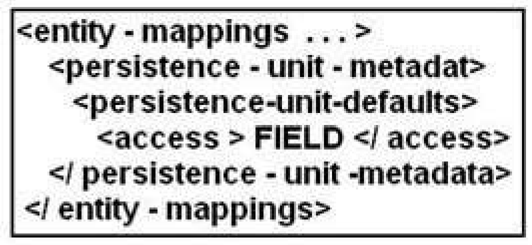
```
public class Employee {
@Id
@Column(name="EMP_ID")
private long id;

@ElementCollection
@CollectionTable(
name="PHONE",
joinColumns=@JoinColumn(name="OWNER_ID")
)
private List<Phone> phones;

}
@Embeddable
public class Phone {
private String type;
private String areaCode;
@Column(name="P_NUMBER")
private String number;
}
```

References:

QUESTION 4
Consider a persistence application with the following orm.xml:



```
<entity - mappings  . . . >
   <persistence - unit - metadat>
     <persistence-unit-defaults>
        <access > FIELD </ access>
   </ persistence - unit -metadata>
 </ entity - mappings>
```

What will be the effect of the above orm.xml?

A. The access type for only those entities that have not explicitly specified @Access will be defaulted to field.
B. The access type for all entities in the persistence unit will be changed to FIELD.
C. The access type for all entities specified in this orm.xml will be changed to FIELD.

D. The access type for only those entities defined in this orm.xml for which access is not specified will be defaulted to FIELD.

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Set the default to FIELD-access.

Note:persistence-unit-defaults
Rule:Full override
Description:If a schema setting exists, then the eclipselink-orm.xml schema setting overrides the existing setting or creates a new schema setting.

QUESTION 5
An entity person is mapped to a table PERSON and has a collection-valued persistence field otherUsedNames that stores names used by a person. The otherUsedNames field is mapped to a separate table called NAMES. Which code fragment correctly defines such field?

A. @ElementCollection (name = "NAMES")Protected set<String> otherUsedNames = new HashSet () ;
B. @ElementCollection@ElementTable (name = "NAMES")Protected set<String> otherUsedNames = new HashSet () ;
C. @ElementCollection@SecondaryTable (names = "NAMES")Protected set<String> otherUsedNames = new HashSet () ;
D. @ElementCollection@CollectionTable (names = "Names")Protected set<String> otherUsedNames = new HashSet () ;

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
CollectionTableSpecifies the table that is used for the mapping of collections of basic or embeddable types. Applied to the collection-valued field or property.

Example:@Entity public class WealthyPerson extends Person { @ElementCollection @CollectionTable(name="HOMES") // use default join column name

References:

QUESTION 6
Which statement is true about the @OrderColumn annotation?

A. If mime is not specified, it defaults to the foreign key column.
B. The OrderColumn annotation may be specified only on a relationship.
C. The OrderColumn annotation is always specified on the owning side of a relationship.
D. The order column is not visible as part of the state of the entity.

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:

The order column is not visible as part of the state of the entity or embeddable class.

References:

QUESTION 7
Which cascade option can be specified in a mapping descriptor so that it applies to all relationships in a persistent e unit?

A. cascade all
B. cascade detach
C. cascade remove
D. cascade-persist

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
cascade-persist
The cascade-persist subelement applies to all relationships in the persistence unit. Specifying this subelement adds the cascade persist option to all relationships in addition to any settings
specified in annotations or XML.

References:

QUESTION 8
The developer wants to define a unidirectional relationship from the customer entity to the order entity and map this relationship using a foreign key mapping strategy.

Which one of the pairs of classes below correctly achieves this task?

A. @Entity public class Customer {@Id int customerId;@OneToMany @JoinColumn (name = "CUST_ID") Set <Order> orders;. . .}@Entity public class order {@Id int orderId;. . .}
B. @Entity public class Customer {@Id int customerId;@OneToMany Set <Order> orders;. . .} @Entity@JoinColumn (names = "CUST-ID", referencedColumnName = "customerId")public class order {@Id int order Id;. . .}
C. @Entity public class Customer {@Id int customerId;@OneToMany (JoinColumn = @joinColumn (name = "CUST_ID") Set <Order> orders;. . .}@Entity public class order {@Id int orderId;. . .}
D. @ Entity public class Customer {@Id int customerId;@OneToMany (JoinColumn = @JoinColumn (name = "CUST_ID"), table = ""ORDER) Set <Order> orders;. . .}@Entity public class order {@Id int orderId;. . .}

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
JoinColumnspecifies a column for joining an entity association or element collection.

Example: unidirectional one-to-many association using a foreign key mapping

```
// In Customer class
@OneToMany
@JoinColumn(name="CUST_ID") // join column is in table for Order public Set<Order> getOrders() {return orders;}
```

References:

QUESTION 9
A developer who is designing entity classes to map a legacy database encounters a table called STUDENT_RECORD.

This table has two columns, STUDENT_ID and STUDENT_INFO_ID. The primary key of this table consists of both columns, and there is a unique constraint on each info column.

The STUDENT_ID column is foreign key to the STUDENT table and STUDENT_INFO_ID column is a foreign key to the STUDENT_DAT table.

What entity classes and relationships can the developer use to model these tables and relationship? (Choose two)

A. Model the student table as a student entity and STUDENT_DATA table StudentData entity. Model the STUDENT_RECORDS table as bidirectional many-to-many relationship between the student entity on the student data entity.
B. Model the STUDENT table as a Student entity and the STUDENT-DATA table as a StudentData entity. Model the STUDENT_RECORD table as a bidirectional one-to-one relationship between the student entity and the StudentData entity.
C. Model the STUDENT table as a Student entity and the STUDENT-DATA table as a StudentData entity. Model the Student-Records table as a student record entity. Create a many-to-one one relationship from the StudentRecord entity to the student entity and many-to-one relationship from the student entity to the StudentData entity and one-to-many relationship from the StudentData entity to the StudentRecord entity.
D. Model the STUDENT table as a Student entity and the STUDENT-DATA table as a StudentData entity. Model the STUDENT-RECORD table as a StudentRecord entity. Create a bidirectional one-to- one relationship between the StudentRecord entity and bidirectional one-to-one relationship between the Student Record entity and the Student Data entity.

Correct Answer: AC
Section: (none)
Explanation

Explanation/Reference:
Explanation:
There is a many-to-many relationship between the Student table and the Student_table. This can be modeled with one many-to-many relation (A) or two many-to-one relations (C).

Incorrect
not B, not D: Not a one-one relationship.

QUESTION 10
The developer is creating a Java Persistence model for a legacy database. The database contains customer and subscriptions.

The subscriptions table has a foreign key to the Customer table, a foreign key to the magazines table, and a column that stores a java.util.Date (the subscription expiration date).

The developer wants to create entities Customer and subscription to model these tables and to represent the relationship between customer and subscription as a java.util.Map.

Which one of the following fields of the Customer entity accomplishes this task?

A. @OneToMany@joinColumn (name = "Customer - FK")Map <Magzine, Data> subscriptions;
B. @ElementCollectionMap <Magzine, Data> subscriptions
C. @OneToMany@JoinTable (name = "Subscriptions")Map <Magzine, Data> subscriptions;
D. @ElementCollection@CollectionTable (name = "subscriptions")Map <Magazine, Data> subscriptions
E. @ElementCollection@CollectionTable (Name = "Subscriptions")@Temporal (TemporalType.DATE)

Map<magazine, date> subscriptions

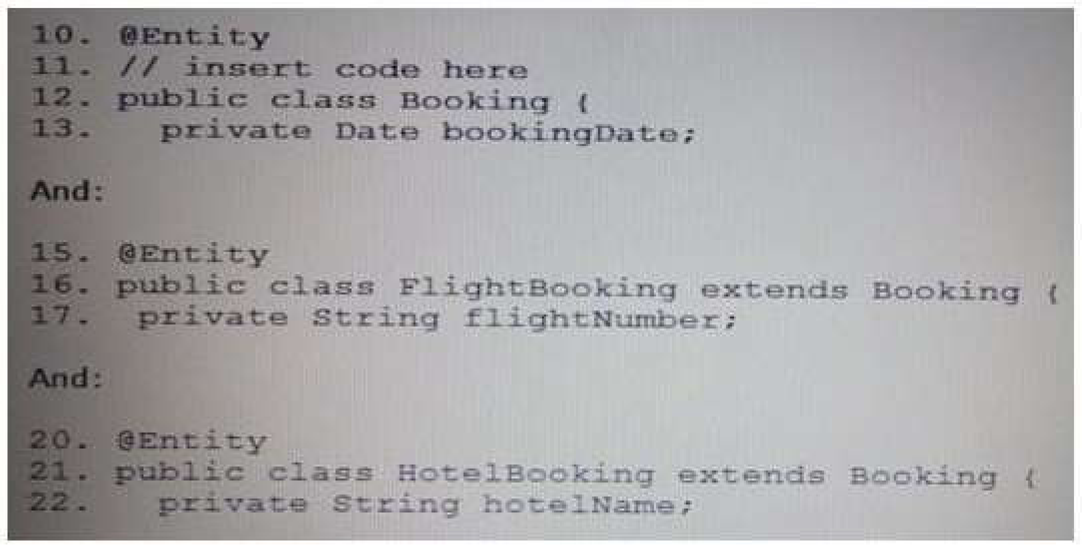
Correct Answer: E
Section: (none)
Explanation

Explanation/Reference:
Explanation:
*The ElementCollection values are always stored in a separate table. The table is defined through the @CollectionTable annotation or the <collection-table> element.

*Annotation Type Temporal
This annotation must be specified for persistent fields or properties of type java.util.Date and java.util.Calendar.

References:

QUESTION 11
Refer to the Exhibit.



```
10. @Entity
11. // insert code here
12. public class Booking {
13.    private Date bookingDate;

And:

15. @Entity
16. public class FlightBooking extends Booking {
17.   private String flightNumber;

And:

20. @Entity
21. public class HotelBooking extends Booking {
22.    private String hotelName;
```

A developer wants to have bookingdata stored in the table BOOKING, flightnumber in table FLIGHTBOOKING, and hotel name in HOTELBOOKING.

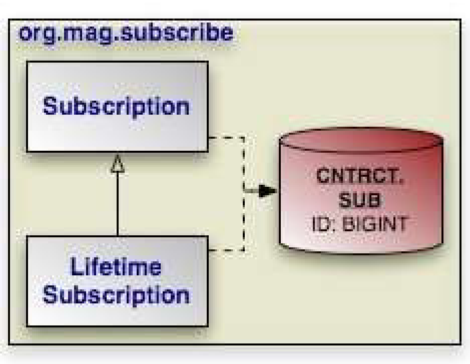
Which code, inserted at line 11 of class Booking, is appropriate for his strategy?

A. @Joined
B. @SingleTable
C. @TablePerClass
D. @Inheritance (strategy = JOINED)
E. @Inheritance (strategy = SINGLE_TABLE)
F. @Inheritance (strategy = TABLE_PER_CLASS)

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
The InheritanceType.SINGLE_TABLE strategy maps all classes in the hierarchy to the base class' table.

Example:



In our model, Subscription is mapped to the CNTRCT.SUB table. LifetimeSubscription, which extends Subscription, adds its field data to this table as well.

```
@Entity
@Table(name="SUB", schema="CNTRCT")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
public class Subscription {

}

@Entity(name="Lifetime")
public class LifetimeSubscription
extends Subscription {

}
```

References:

QUESTION 12
Which two of the following statements are true of embeddable classes? (Choose two)

A. An embeddable class must not be used to represent the state of another embeddable class.
B. Null comparison operations over embeddable classes are not supported in the Java Persistence query language.
C. An embeddable class must not contain a relationship to an entity.
D. An embeddable class can be the key of a Map relationship.

Correct Answer: BD
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Not A:Embeddable classes may themselves use other embeddable classes to represent their state.They may also contain collections of basic Java programming language types or other embeddable classes. Not C:Embeddable classes may also contain relationships to other entities or collections of entities. If the embeddable class has such a relationship, the relationship is from the target entity or collection of entities to the entity that owns the embeddable class.

Note:Embeddable classes are used to represent the state of an entity but don't have a persistent identity of their own, unlike entity classes. Instances of an embeddable class share the identity of the entity that owns it. Embeddable classes exist only as the state of another entity. An entity may have single-valued or collection-valued embeddable class attributes.

References:

QUESTION 13
The department entity has a unidirectional OneToMany relationship to the employee entity. The developer wants to model this relationship as a java.util.map such that the key of map is true employee name. The primary key of the Employees entity is empId, an integer.

Which of the following is correct?

A. @OneToMany (targetEntity = Employee.class)@MapKeyClass (string.class)map employees;
B. @OneToMany @mapKey (name = "name") map < integer, Employee> Employees;
C. @OneToMany @MapKeyJoinColumn (name = "name") map <String, Employee> employees;
D. @OneToMany @mapsId (name = "name") map <String, Employee> employees;

Correct Answer: B
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Annotation Type MapKeyspecifies the map key for associations of type java.util.Map when the map key is itself the primary key or a persistent field or property of the entity that is the value of the map.

Example:

```
@Entity
public class Department {

@OneToMany(mappedBy="department")
@MapKey(name="name")
public Map<String, Employee> getEmployees() {... }

}
@Entity
public class Employee {
@Id public Integer getEmpId() { ... }

@ManyToOne
@JoinColumn(name="dept_id")
public Department getDepartment() { ... }

}
```

References:

QUESTION 14
The developer wants to override the default mappings for an embeddable class Address used by the customer entity.

The Address class is defined as follows:

```
@Embeddable public class Address (
private String street;
private String city;
private String country;
. . .
)
```

Assume that NO mapping descriptor is present. Which code segment below shows the correct way to override the default mapping for address?

A. @AttributeOverrides ({@AttributeOverride (name = "street", column = @Column (name = ADDR_STREET)),@AttributeOverride (name = "city, column = @Column (name = ADDR_CITY)),@AttributeOverride (name = "country, column = @Column (name = ADDR_COUNTRY)),}} @Embedded Address addr;
B. @ AttributeOverrides ({@AttributeOverride (name = "street", column = @Column (name = "name_STREET")),@AttributeOverride (name = "city, column = @Column (name = "name_CITY")),@AttributeOverride (name = "country, column = @Column (name = "name_COUNTRY")),}} @Embedded Address addr;
C. @ AttributeOverrides ({@AttributeOverride (name = "street", column (name = "name_STREET")),@AttributeOverride (name = "city, column (name = "name_CITY")),@AttributeOverride (name = "country, column (name = "name_COUNTRY")),}}@Embedded Address addr;
D. @AttributeOverrides ({@AttributeOverride (name = "addr.street", column = @Column (name = ADDR_STREET)),@AttributeOverride (name = "addr.city", column = @Column (name = ADDR_CITY)),@AttributeOverride (name = "addr.country", column = @Column (name = ADDR_COUNTRY)),}}@Embedded Address addr;

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
*Annotation Type Embedded
Defines a persistent field or property of an entity whose value is an instance of an embeddable class.
Here we have @Embedded Address addr;

*AttributeOverridesis used to override mappings of multiple properties or fields.

Example:

```
@Embedded
@AttributeOverrides({
@AttributeOverride(name="startDate", column=@Column("
EMP_START")),
@AttributeOverride(name="endDate", column=@Column("EMP_END")) })
public EmploymentPeriod getEmploymentPeriod() { ... }
```

References:

QUESTION 15
The Contact Information embeddable class contains address information as well as a reference to a phone entity. The ContactInformation class is defined as follows:

```
@Embeddable public class ContactInformation {

String street;
String city;
@OneToOne Phone phone;
}
```

The developer wants to use this class in an Employee entity, but override the default name of the foreign key to the Phone entity. Which of the code segments shows how to do this correctly?

A. @Entity public class Employee {@Id int empId;@AssociationOverride (name = empInfo.phone", joinColumn = @JoinColumn)(name = "INFO_FK"))ContactInformation empInfo;}

B. @AssociationOverride (name = "empInfo.phone", joinColumn = "INFO_FK")@Id int empId;@ContactInformation empInfo;}
C. @ AssociationOverride (name = "empInfo.phone", joinColumn = "INFO_FK")Entity public class Employee {@Id int empId;}
D. Entity public class Employee {@Id int empId;@ AssociationOverride (name = "empInfo.phone", joinColumn = "INFO_FK")ContactInformation empInfo;}

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Example:
Example 2: Overriding the mapping for phoneNumbers defined in the ContactInfo class

```
@Entity
public class Employee {
@Id int id;
@AssociationOverride(
name="phoneNumbers",
joinTable=@JoinTable(
name="EMPPHONES",
joinColumns=@JoinColumn(name="EMP"),
inverseJoinColumns=@JoinColumn(name="PHONE")
)
)
@Embedded ContactInfo contactInfo;

}
```
References:

QUESTION 16
A stateless session bean's business method invokes EJBContext.setRollBackOnly and receives an IllegalStateException.

Under which of these conditions could this be possible?

A. The business method is marked with the MANDATORY transaction attribute.
B. The business method is marked with the NONSUPPORTED transaction attribute.
C. This Is NOT possible; a stateless session bean cannot invoke EJBContext.SetRollBackOnly.
D. The bean has no metadata (in annotations 01 deployment descriptor) which specifies the transaction attribute for the method.

Correct Answer: B
Section: (none)
Explanation

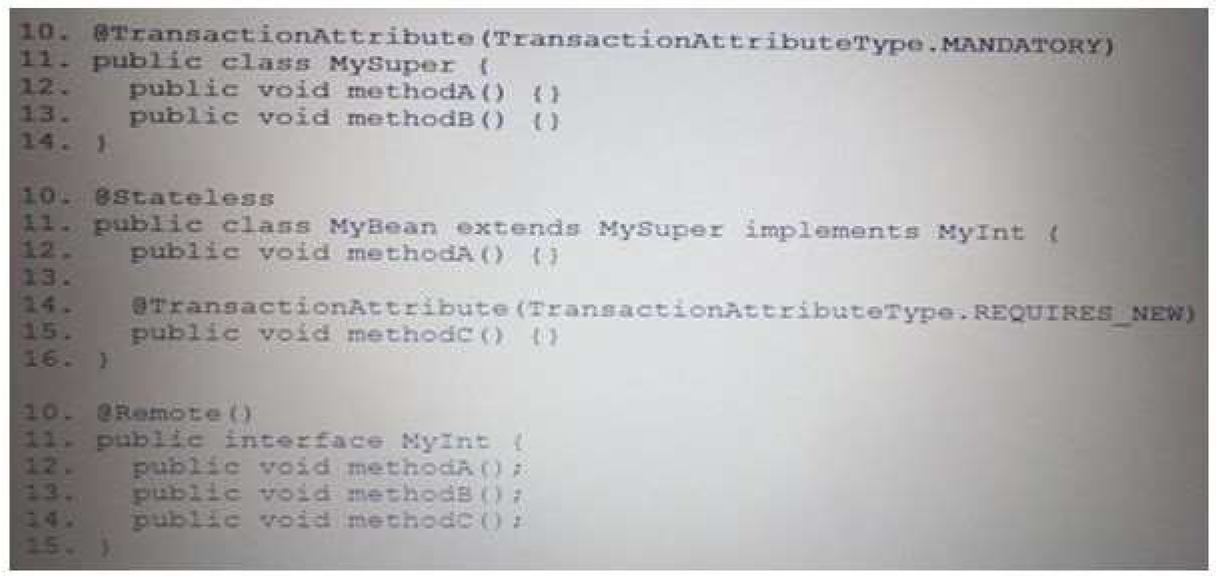
Explanation/Reference:
Explanation:
setRollbackOnly

void setRollbackOnly()
throws java.lang.IllegalStateException
Mark the current transaction for rollback. The transaction will become permanently marked for rollback. A transaction marked for rollback can never commit. Only enterprise beans with container- managed transactions are allowed to use this method.

Throws:
IllegalStateException - The Container throws the exception if the instance is not allowed to use this method (i.e. the instance is of a bean with bean-managed transactions).

References:

QUESTION 17
Given the following stateless session bean implementation classes:



```
@TransactionAttribute(TransactionAttributeType.MANDATORY)
public class MySuper {
  public void methodA() {}
  public void methodB() {}
}

@Stateless
public class MyBean extends MySuper implements MyInt {
  public void methodA() {}

  @TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
  public void methodC() {}
}

@Remote()
public interface MyInt {
  public void methodA();
  public void methodB();
  public void methodC();
}
```

Assuming no other transaction-related metadata, what are the transaction attributes on methodB, and method C respectively?

A. MANDATORY, MANDATORY, and MANDATORY
B. REQUIRED, MANDATORY, and REQUIRES_NEW
C. MANDATORY, MANDATORY,and REQUIRES__NEW
D. REQUIRED, REQUIRES_NEW, and REQUIRES_NEW

Correct Answer: B
Section: (none)
Explanation

Explanation/Reference:
Explanation:
* TransactionAttributeType
The enum TransactionAttributeType is used with the TransactionAttribute annotation to specify whether the methods of a session bean or message driven bean are called with a valid transaction context.

References:

QUESTION 18
A session bean business method throws an exception during execution.

Which two are responsibilities of the Bean Provider when throwing the exception? (Choose two.)

A. For application exceptions, ensure that if the current transaction commits there will be no loss of data integrity.
B. For application exceptions, ensure that the current transaction will commit.

C. For system errors, when the client is remote through a java.rmi.remoteException that wraps the original exception.
D. For checked exceptions from which the bean cannot recover, throw an EJBException that wraps the original exception.

Correct Answer: AD
Section: (none)
Explanation

Explanation/Reference:
Explanation:
D:If the bean method performs an operation that results in a checked (i.e. not a subclass of java.lang.RuntimeException) exception that the bean method cannot recover, the bean method should throw the javax.ejb.EJBException that wraps the original exception.

A:The Bean Provider MUST do one of the following to ensure data integrity before throwing an application exception from an enterprise bean instance:

*Ensure that the instance is in a state such that a client's attempt to continue and/or commit the transaction does not result in loss of data integrity. For example, the instance throws an application exception indicating that the value of an input parameter was invalid before the instance performed any database updates.

*If the application exception is not specified to cause transaction rollback, mark the transaction for rollback using the EJBContext.setRollbackOnly method before throwing the application exception. Marking the transaction for rollback will ensure that the transaction can never commit.
References:

QUESTION 19
FooBean and BarBean are both EJB 3.0 stateless beans with container-managed transaction demarcation. All business methods in FooBean have transaction attribute REQUIRED, and all business methods in BarBean have transaction attribute REQUIRED_NEW. The business method foo in FooBean invokes the Business method bar in BarBean.

Given:
```
10. Public class BarBean {
11. public void bar () {
12. throw new RuntimeException ("unexpected error . . . ");
13. }}
```

Which is true about the method of invocation assuming execution reaches line 12?

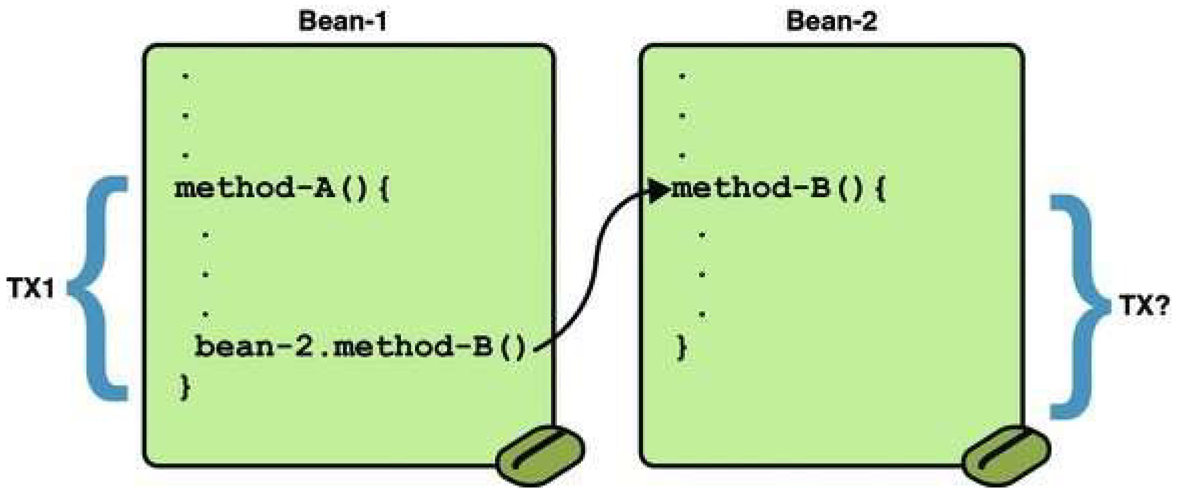
A. FooBean.foo method receives javax.ejb.EJBException.
B. The BarBean bean instance is in ready state for the next invocation.
C. FooBean.foo method receives javax -ejb. EJBTtansactionRolledbackException.
D. FooBean.foo method receives the original RuntimeException thrown from BarBean.bar method.

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
The exception will be thrown within the FooBean container.
A standard EJBException will be thrown.
The EJBException is thrown to report that the invoked business method or callback method could not be completed because of an unexpected error (e.g. the instance failed to open a database connection).

Note:

Transaction Attributes
A transaction attribute controls the scope of a transaction. Figurebelowillustrates why controlling the scope is important. In the diagram, method-A begins a transaction and then invokes method-B of Bean-2. When method-B executes, does it run within the scope of the transaction started by method-A, or does it execute with a new transaction? The answer depends on the transaction attribute of method-B.



*RequiresNew AttributeIf the client is running within a transaction and invokes the enterprise bean's method, the container takes the following steps:
If the client is not associated with a transaction, the container starts a new transaction before running the method.
You should use the RequiresNew attribute when you want to ensure that the method always runs within a new transaction.

References:

QUESTION 20
Given a set of CMT bean methods with the following transaction attributes:

Method M1 = SUPPORTS
Method M2 = REQUIRED
Method M3 = NOT_SUPPORTED
Method M4 = REQUIRES_NEW

And the following method invocation sequence:

Method M1 invokes Method M2
Method M2 invokes Method M3
Method M1 invokes Method M4

If Method M1 is invoked by a method that does NOT have a transaction context, which describes a possible scenario?

A. Method M1:notransactionMethodM2:newtransactionMethodM3:notransactionMethodM4:newtransaction
B. MethodM1:notransactionMethodM2:ContainerthrowsTransactionNotSupportedException
C. MethodM1:notransactionMethodM2:runsinsametransactionasM1MethodM3:containerthrowsTransactionNotSupportException
D. MethodM1:notransactionMethodM2:newtransactionMethodM3:ContainerthrowsTransactionNotSupportException.

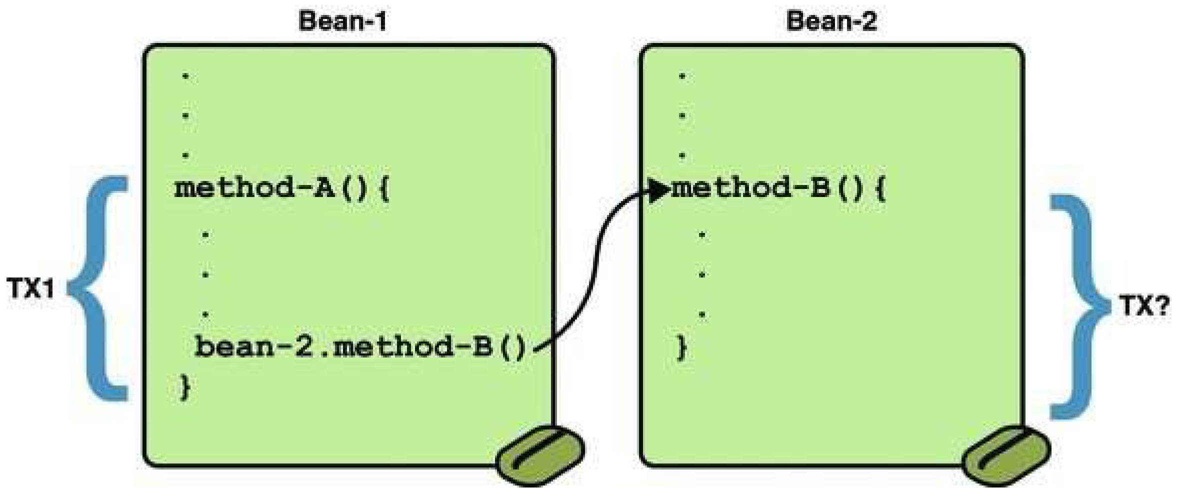
Correct Answer: A
Section: (none)

Explanation

Explanation/Reference:
Explanation:
M1 has no transaction.
As M2 has Required attribute it will run within a new transaction.

Note:
*Required AttributeIf the client is running within a transaction and invokes the enterprise bean's method, the method executes within the client's transaction. If the client is not associated with a transaction, the container starts a new transaction before running the method. The Required attribute is the implicit transaction attribute for all enterprise bean methods running with container-managed transaction demarcation. You typically do not set the Required attribute unless you need to override another transaction attribute. Because transaction attributes are declarative, you can easily change them later.

Transaction Attributes
A transaction attribute controls the scope of a transaction. Figurebelowillustrates why controlling the scope is important. In the diagram, method-A begins a transaction and then invokes method-B of Bean-2. When method-B executes, does it run within the scope of the transaction started by method-A, or does it execute with a new transaction? The answer depends on the transaction attribute of method-B.



References:

QUESTION 21
WhichEntityManager API will lock entity with a pessimistic lock?

A. em.lock(x,LockModeType.WRITE)
B. em.lock(x,LockModeType.PESSIMISTIC)
C. em.lock(x,LockModeType.PESSIMISTIC_READ)
D. em.lock(x,LockModeType.OPTIMISTIC_FORCE_INCREMENT)

Correct Answer: C
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Setting a Pessimistic Lock
An entity object can be locked explicitly by the lock method:

em.lock(employee, LockModeType.PESSIMISTIC_WRITE);

The first argument is an entity object. The second argument is the requested lock mode.

A LockTimeoutException is thrown if the requested pessimistic lock cannot be granted:

*A PESSIMISTIC_READ lock request fails if another user (which is represented by another EntityManager instance) currently holds a PESSIMISTIC_WRITE lock on that database object.

*A PESSIMISTIC_WRITE lock request fails if another user currently holds either a PESSIMISTIC_WRITE lock or a PESSIMISTIC_READ lock on that database object.
References:

QUESTION 22
It a Persistence application locks entity x with a pessimistic lock, which statement is true?

A. Persistence provider can defer obtaining the lock until the next synchronization of an entity to the database.
B. A Persistence provider will obtain the lock when the entity is refreshed from the database.
C. A Persistence provider is not required to support the LockModeType. PESSIMISTIC_WRITE locking type.
D. If a lock cannot be obtained, the Persistence provider must throw an exception.

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Explanation: When the lock cannot be obtained, and the database locking failure results in transaction- level rollback, the provider must throw the PessimisticLockException and ensure that the JTA transaction or EntityTransaction has been marked for rollback.

References:

QUESTION 23
If a Persistence application locks entity x with a LockModeType.OPTIMISTIC_FORCE INCREMENT lock type, which statement is true?

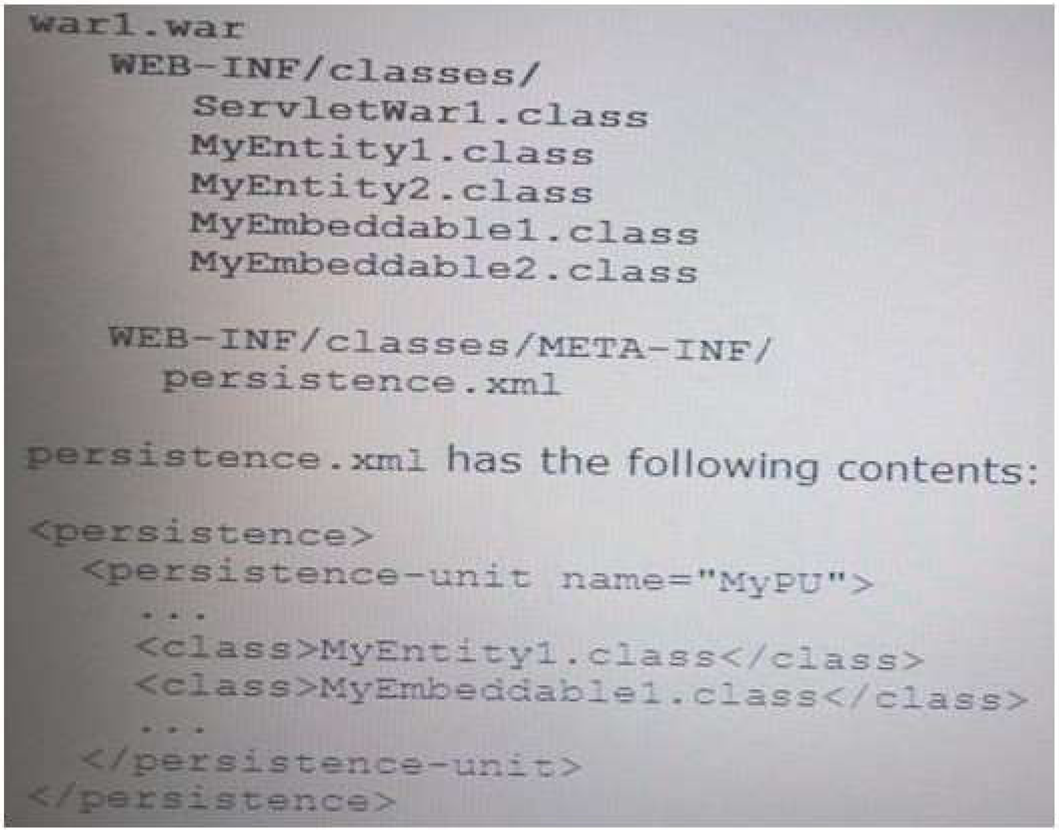
A. The Persistence application must increment the version value prior to locking the entity.
B. This operation will result in a PersistentLockException for a non-versioned object.
C. This operation will result in a PersistentLockException if the version checks fail.
D. LockModeType.OPTIMISTIC_FORCE_INCREMENT is the synonym of the LockModeType.WRITE lock type.

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Lock ModeWRITE is asynonym for OPTIMISTIC_FORCE_INCREMENT. Use of LockModeType.OPTIMISTIC_FORCE_INCREMENT is to be preferred for new applications.

References:

QUESTION 24
Consider the following web application:



```
war1.war
    WEB-INF/classes/
        ServletWar1.class
        MyEntity1.class
        MyEntity2.class
        MyEmbeddable1.class
        MyEmbeddable2.class

    WEB-INF/classes/META-INF/
      persistence.xml
```

persistence.xml has the following contents:

```
<persistence>
  <persistence-unit name="MyPU">
    ...
    <class>MyEntity1.class</class>
    <class>MyEmbeddable1.class</class>
    ...
  </persistence-unit>
</persistence>
```

Here MyEntity1.class and MyEntity2.class are annotated with @Entity and MyEmbeddable1-class and MyEmbeddable2-class are annotated with @Embeddable. MyPU is container managed. Which of the following represents set of classes considered managed by MyPU?

A. MyEntity1, and MyEmbeddable1
B. MyEntity1, MyEmbeddable1, MyEntity2, and MyEmbeddable2
C. MyEntity1, MyEmbeddable1, and MyEntity2
D. MyEntity1 and MyEntity2

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Persistence unit
The set of entity types that can be managed by a given entity manager is defined by a persistence unit. A persistence unit defines the set of all classes that are related or grouped by the application, and which must be collocated in their mapping to a single data store.

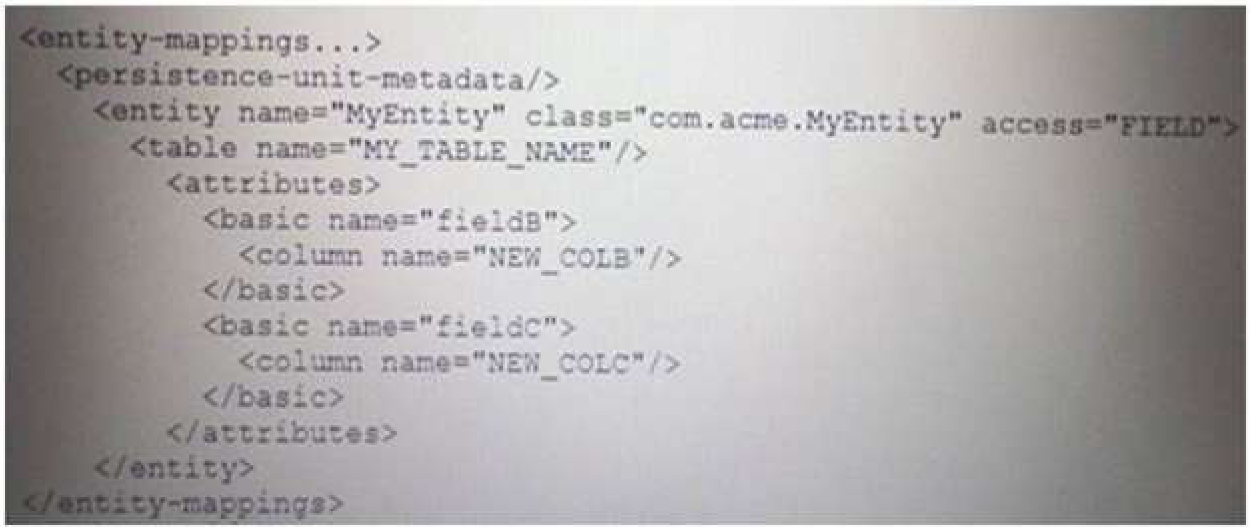
QUESTION 25
Consider a persistence application with following entity:

@Entity
Public class MyEntity {

@Column (name = "FIELDA_COLUMN")
int fieldA;
int fieldB;

int fieldC;
int fieldD;

An orm.xml is packaged in the application with the following contents:



```
<entity-mappings...>
  <persistence-unit-metadata/>
    <entity name="MyEntity" class="com.acme.MyEntity" access="FIELD">
      <table name="MY_TABLE_NAME"/>
        <attributes>
          <basic name="fieldB">
            <column name="NEW_COLB"/>
          </basic>
          <basic name="fieldC">
            <column name="NEW_COLC"/>
          </basic>
        </attributes>
    </entity>
</entity-mappings>
```

Which two of following statement are true. (Choose two)

A. fieldA is mapped to column FIELDA
B. fieldB is mapped to column NEW_COLB
C. fieldD is a persistent attribute of MyEntity
D. fieldD is not a persistence attribute of MyEntity

Correct Answer: BC
Section: (none)
Explanation

Explanation/Reference:
Explanation:
B: We have_
<basic name="fieldB">
<column name="NEW_COLB">

C:Persistent Fields
Every non-static non-final entity field is persistent by default unless explicitly specified otherwise (e.g. by using the @Transient annotation).

References:

QUESTION 26
A developer wants to write a type-safe Criteria API query. Which two of the following statements true about Criteria query roots? (Choose two)

A. The query MUST define a query root.
B. The query MUST define a query root only if it navigates to related entities.
C. The query MUST NOT define multiple query roots.
D. The query may define multiple query roots.

Correct Answer: BD

Section: (none)
Explanation

Explanation/Reference:
Explanation:
B:Query Roots
For a particular CriteriaQueryobject, the root entity of the query, from which all navigation originates, is called the query root. It is similar to the FROM clause in a JPQL query.

D:Criteria queries may have more than one query root. This usually occurs when the query navigates from several entities.

The following code has two Root instances:

```
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet1 = cq.from(Pet.class);
Root<Pet> pet2 = cq.from(Pet.class);
```

References:

QUESTION 27
Which of the following Criteria query snippets demonstrates the correct way to create and execute strongly typed queries? Assume that cb references an instance of the CriteriaBuilder interface and em references an EntityManager instance.

A. CriteriaQuery <office> cq = cb.createQuery (Office.class);. . .TypedQuery<Office> tq = em.createQuery (cq);List <office> offices =cb.getResultList ();
B. CriteriaQuery cq = cb.createQuery (Office.class). . .TypedQuery<office> tq = em.createQuery (cq, office.class);List <office> offices = tq.getresult ();
C. CriteriaQuery<office> cq = em.createQuery (cq, office.class);. . .TypedQuery<Office> tq = em.createQuery (cq);List <office> offices = tq.getresult ();
D. CriteriaQuery <office> cq = cb.createQuery (Office.class);. . .TypedQuery<Office> tq = em.createQuery (cq);List<office> Offices = tq.getResultList ();

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Example 1: (not B, not C)
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);

Example 2: (not A)
The following query retrieves all the Country objects in the database. Because multiple result objects are expected, the query should be run using the getResultList method:

```
TypedQuery<Country> query =
em.createQuery("SELECT c FROM Country c", Country.class); List<Country> results = query.getResultList();
```

References:

QUESTION 28
A developer is writing an application with three java Persistence API entities: order, customer, and Address. There is a many-to-one relationship between order and customer, and a one to-many relationship between customer and Address.

Which two Criteria queries will return the orders of all customers who have an address whose value specified by the String parameter postalcode? (Choose two)

A. String postalCode = . . .Criteria Builder cb = . . .CriteriaQuery<order> cq = cb.createQuery (Order.class);Root <order> order = cq.from(order.class);Join <order, Customer> customer = order.join (Order_.customer);Root <Order> order = cq.from (Order.class);Join <customer, Address> address = customer join (Order_.customer)cq.where (cb.equal (address.get(Address_.postalCode), postalCode));cq.select (order). Distinct (true);// query execution code here. . .
B. String postalCode = . . .Criteria Builder cb = . . .Root <Order> order = cq.from (Order.class);order.join (order_. customer).join(Customer_.addresses);cq.where (cb.equal (address.get(Address_.postalCode), postalCode));cq.select (order). Distinct (true);// query execution code here
C. String postalCode = ...CriteriaBuilder cb = ...Root<order> order = cq.from (Order . class) ,Join<order, Address> address = order.join(Customer_.addresses);cq.where(ct>.equal(address.get(Address_.postalCode), postalCode));cq-select(order).distinct(true);// query execution code here. . .
D. String postalCode = ...CriteriaBuilder cb = ...Root<order> order = cq.from (Order . class) ,Join<order, Address> address = order . join (Order_. customer).join (Customer_.addresses);cq.where <cb.equal (address.get(Address_.postalCode) , postalCode)) ; cq.selec:(order).distinct(true);// query execution code here. . .

Correct Answer: AD
Section: (none)
Explanation

Explanation/Reference:
Explanation:
A: Join Order and Customer and join Customer and Address. Works fine.
Not B: Chained joined not set up correctly.
C: Join Order and Address through first joining Order and Customer. Not D: Cannot Join Order Address it just one single join. Need to chain the join.

Note:Querying Relationships Using Joins
For queries that navigate to related entity classes, the query must define a join to the related entity by calling one of the From.join methods on the query root object or another join object. The join methods are similar to the JOIN keyword in JPQL.

The target of the join uses the Metamodel class of type EntityType<T> to specify the persistent field or property of the joined entity.

The join methods return an object of type Join<X, Y>, where X is the source entity and Y is the target of the join. In the following code snippet, Pet is the source entity, Owner is the target, and Pet_ is a statically generated metamodel class:

```
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);

Root<Pet> pet = cq.from(Pet.class);
Join<Pet, Owner> owner = pet.join(Pet_.owners);
```

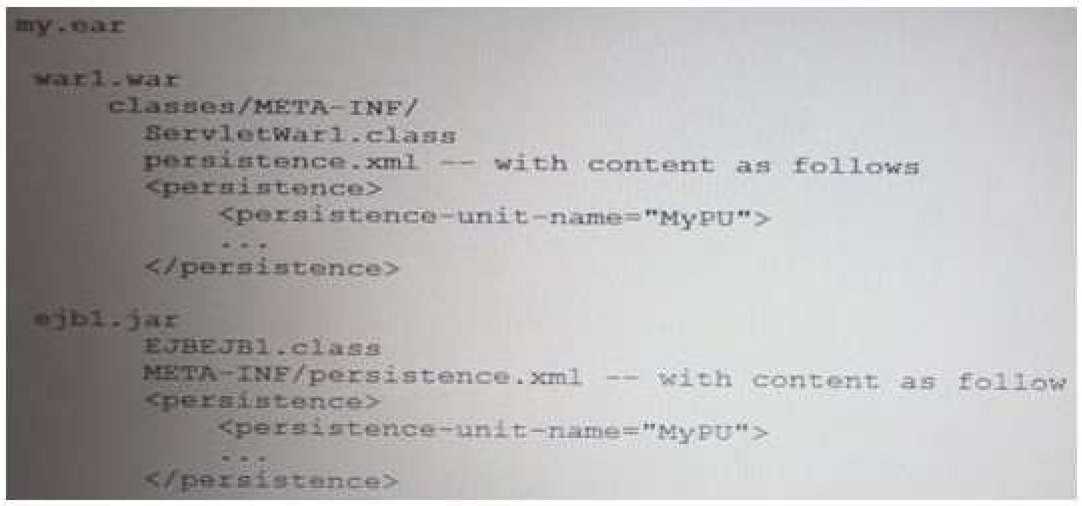
Joins can be chained together to navigate to related entities of the target entity without having to create a Join<X, Y> instance for each join:

```
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);

Root<Pet> pet = cq.from(Pet.class);
Join<Owner, Address> address = cq.join(Pet_.owners).join(Owner_.addresses);
```

References:

QUESTION 29
A JavaEE application is packaged as follows.



```
my.ear

 war1.war
      classes/META-INF/
        ServletWar1.class
        persistence.xml -- with content as follows
        <persistence>
            <persistence-unit-name="MyPU">
            ...
        </persistence>

 ejb1.jar
        EJBEJB1.class
        META-INF/persistence.xml -- with content as follow
        <persistence>
            <persistence-unit-name="MyPU">
            ...
        </persistence>
```

Which of the following is true for a portable JavaEE application?

A. This is an invalid application. A JavaEE application cannot have more than one persistent with same name.
B. "MyPu" defined under each module is visible to only the defining module. There is no way other modules can access it.
C. Code in the ejb1.jar can access "MyPU" defined under war1.war using "war1#myPU"

Correct Answer: B
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Persistence units are defined by the persistence.xml configuration file. The JAR file or directory whose META-INF directory contains persistence.xml is called the root of the persistence unit. The scope of the persistence unit is determined by the persistence unit's root. Each persistence unit must be identified with a name that is unique to the persistence unit's scope.

References:

QUESTION 30
The developer has modeled student interests as a set <String>:

```
@Entity public class Student {
@Id int student Id;
string name;
@ElementaryCollection
Set <String> Interests;
. . .
}
```

The developer wants the values of this set to be stored using a column named student_interests.
Select the item below that accomplishes this task:

A. @ElementaryCollection@Column(name = "student_interests")Set <string> interests;
B. @ElementaryCollection (column = "student_intersets") Set<String> interests;
C. @ElementaryCollection @CollectionTable (column = "student_intersets") Set<String> interests;
D. @ElementaryCollection @CollectionTable (column = @column(name = "student_interests")) Set <String> interests;

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Example of a ElementCollection relationship to a basic value annotations[edit] @Entity

```
public class Employee {
@Id
@Column(name="EMP_ID")
private long id;

@ElementCollection
@CollectionTable(
name="PHONE",
joinColumns=@JoinColumn(name="OWNER_ID")
)
@Column(name="PHONE_NUMBER")
private List<String> phones;

}
```

References:

QUESTION 31
The embeddable class ContractInformation is used in an element collection of the Employee entity.

```
@Entity
Public class Employee {

@Id int empId;
@ElementaryCollection Set <ContractInformation> info;
. . .
}
```

Assume that the phone class is an entity and that address is an embedded class.

Which two of the code segments below can be used to model the state of ContractInformation? (Choose two)

A. @OneToMany Set <phone> phones;
B. @Embeddable Address address;
C. @ManyToOne phone phone;
D. @ElementaryCollection <Phone> phones;
E. @OneToOne Address address;

Correct Answer: BC
Section: (none)
Explanation

Explanation/Reference:
Explanation:
B:Embeddable classes have the same rules as entity classes but are annotated with the javax.persistence.Embeddable annotation instead of @Entity.

Example:@EmbeddedZipCode zipCode;

C:@ManyToOnedDefines a single-valued association to another entity class that has many-to-one multiplicity.

Example2:

```
@Entity
public class Employee {
@Id int id;
@Embedded JobInfo jobInfo;

}

@Embeddable
public class JobInfo {
String jobDescription;
@ManyToOne ProgramManager pm; // Bidirectional
}
```

References:

QUESTION 32
A developer wants to create a Java Persistence query that will include a subquery.

Which three are true? (Choose three.)

A. Subqueries can be used in a FROM clause.
B. Subqueries can be used in a WHERE clause.
C. The ANY expression can be used only with a subquery.
D. The EXISTS expression can be used only with a subquery.
E. The MEMBER expression can be used only with a subquery.

Correct Answer: BCD
Section: (none)
Explanation

Explanation/Reference:
Explanation:
B (not A):Subqueries can be used in the Criteria API in the select, where, order, group by, or having clauses.
CD:A subQuery is created from a CriteriaQuery using the subQuery operation. Most subQuery usage restricts the subQuery to returning a single result and value, unless used with the CriteriaBuilder exists, all, any, or some operations, or with an in operation.

References:

QUESTION 33
Which statement is correct about the Java Persistence API support for the SQL queries?

A. SQL queries are NOT allowed to use parameters.
B. The result of an SQL query is not limited to entities.
C. Only SELECT SQL queries are required to be supported.
D. SQL queries are expected to be portable across databases.

Correct Answer: C
Section: (none)
Explanation

Explanation/Reference:
Explanation:
The Java Persistence Query Language (JPQL) is a platform-independent object-oriented query language defined as part of the Java Persistence API (JPA) specification.

In addition to retrieving objects (SELECT queries), JPQL supports set based UPDATE and DELETE queries.

Incorrect:
Not B:EntityManager API offers methods for creating instances of Query for executing native SQL statements. The most important thing to understand about native SQL queries created with EntityManager methods is that they, like JPQL queries, return entity instances, rather than database table records.

References:

QUESTION 34
A user entity is in a one-to-many relationship with a Book entity. In other words, a developer reach the collection of books that a user instance myUser has by using the path expression-"myuser -books".

A developer wants to write a Java Persistence query that returns all users that have only two books.

Which two are valid queries that return this information? (Choose two.)

A. SELECT u FROM User U WHERE SIZE (u.books) = 2
B. SELECT u FROM User WHERE COUNT (u.books) = 2
C. SELECT u FROM User u (WHERE COUNT (b) FROM u.books b) = 2
D. SELECT u FROM user u WHERE (SELECT SIZE (b) FROM u.books b) = 2

Correct Answer: AC
Section: (none)
Explanation

Explanation/Reference:
Explanation:
A (not D):The SIZE(collection) function returns the number of elements in a specified collection.

For example:

SIZE(c.languages) is evaluated to the number of languages in that collection.

C: Example(the aggregate query returns a Long):
Query q2 = em.createQuery(
"SELECT COUNT(e) FROM Employee e WHERE e.salary > :limit");

References:

QUESTION 35
A session bean business method invokes UserTransaction.setRollbackonly and receives an IllegalStateException.

Under which circumstance can this happen?

A. The bean is using bean-managed transactions regardless of whether there is an active transaction.
B. There is no circumstance that would cause setRollbackOnly to throw an IllegalStateException.
C. The bean is using bean managed transaction demarcation, and UserTransaccion.begin has been invoked.
D. The setRollbackOnly method is invoked within a bean-managed transaction, and userTransaction.commit has NOT been invoked.

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
SetRollbackOnlythrows anIllegalStateException:

The Container throws the exception if the instance is not allowed to use this method (e.g. if the bean is a stateful session bean)

Note:setRollbackOnlymarksthe current transaction for rollback. The transaction will become permanently marked for rollback. A transaction marked for rollback can never commit. Only enterprise beans with container-managed transactions are allowed to use this method.

References:

QUESTION 36
Persistence application locks entity x with a LockModeType. PESSIMISTIC_READ lock type, which statement is true?

A. This operation will force serialization among transactions attempting to read the entity data.
B. This operation will result in a TransactionRolledbackException if the lock cannot be obtained.
C. If the application later updates the entity, and the changes are flushed to the database, the lock will be converted to an exclusive lock.
D. LockModeType. PESSIMISTIC_READ is the synonym of the LockModeType.READ.

Correct Answer: C
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Lock ModePESSIMISTIC_READ:
Immediately obtain a long-term read lock on the data to prevent the data from being modified or deleted. Other transactions may read the data while the lock is maintained, but may not modify or delete the data.
The persistence provider is permitted to obtain a database write lock when a read lock was requested, but not vice versa.

Note:A lock with LockModeType.PESSIMISTIC_READ can be used to query data using repeatable- read semantics without the need to reread the data at the end of the transaction to obtain a lock, and without blocking other transactions reading the data.

Incorrect:
Not A:A lock with LockModeType.PESSIMISTIC_WRITE, but not withLockModeType.PESSIMISTIC_READ,can be obtained on an entity instance to force serialization among transactions attempting to update the entity data. Not B:If a pessimistic lock cannot be obtained on the database rows, and the failure to lock the data results in a transaction rollback, a PessimisticLockException is thrown. If a pessimistic lock cannot be obtained, but the locking failure doesn't result in a transaction rollback, a LockTimeoutException is thrown.
Not D:Lock ModeREAD is asynonym for OPTIMISTIC.

References:

QUESTION 37
A user entity is retrieved in a query and stored in an instance variable user. The user entity has a single valued name property that uses the mapping defaults, and a photo property, which is lazily loaded. The application then calls the following method:

PersistenceUtil.isLoaded (user);

Which two of the following statements are correct?

A. The name property was loaded from the database.
B. The name property was NOT be loaded from the database.
C. The name property may or may not have been loaded from the database.

D. The photo property was loaded from the database.
E. The photo property was NOT loaded from the database.
F. The photo property may or may not have been loaded from the database.

Correct Answer: AF
Section: (none)
Explanation

Explanation/Reference:
Explanation:
An entity is considered to be loaded if all attributes with FetchType.EAGER--whether explictly specified or by default--(including relationship and other collection-valued attributes) have been loaded from the database or assigned by the application. Attributes with FetchType.LAZY may or may not have been loaded.
References:

QUESTION 38
Given the following entity classes:

```
@Entity
@cacheable (true)
Public class A { . . . }

@Entity
@cacheable (false)
Public class B { . . .}

@Entity
Public class C { . . . }
```

If the shared-cache-mode element of persistence.xml is set to ENABLE _SERVICE, which entities are cached when using a persistence provider that supports caching?

A. A only
B. A and B
C. A and C
D. A, B, and C

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
ENABLE_SELECTIVE
Caching is enabled for all entities for Cacheable(true) is specified.
References:

QUESTION 39
An application has three entities: the mapped superclass person class entity, and the parent and child entities, which are subclasses of person.

The application has created four entity Instances:

Caching has been enabled in the persistence unit, the persistence provider supports caching, and none of entities have the Cacheable annotation applied, or a cacheable XML element in persistence.xml.

The application executes the following code:

```
Cache cache = . . . ;
Cache.evict(person.class)
Boolean result = cache.contains (Child.class, 400);
```

Assume there is no concurrent activity involving the cache. Which two statements are correct? (Choose two)

A. Only person1 will be removed from cache.
B. Person1, parent1, child1, and child2 will be removed from cache.
C. Result is true
D. Result is false

Correct Answer: BD
Section: (none)
Explanation

Explanation/Reference:
Explanation:
evict(java.lang.Class cls)
Remove the data for entities of the specified class (and its subclasses) from the cache.

References:

QUESTION 40
A developer wants to ensure that an entity's data is up-to-date with regard to the database. Which of the following statements is guaranteed to accomplish this?

A. Call EntityManager.refresh on the entity.
B. Add a cacheable (false) annotation on the entity class.
C. Call EntityManager.find on the entity.
D. Use a named query to retrieve the entity.

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
refresh(java.lang.Object entity, java.util.Map<java.lang.String,java.lang.Object> properties) Refresh the state of the instance from the database, using the specified properties, and overwriting changes made to the entity, if any.

References:

QUESTION 41
An application has two entities, Department and Employee, and there is a one-to-many relationship between them. The application has the following query:

```
SELECT d
FROM Department d LEFT JOIN FETCH d. employees
WHERE d.name = : name
```

After receiving the results of the query, the application accesses the returned department's Employee entities stored in the Department.employees collection-valued attribute.

All caching has been turned off in the application.

Which statement is true?

A. The database will be accessed once during the query execution phase, and once for each Employee entity in Department - employees.
B. The database will be accessed once during the query execution phase ONLY.
C. The database will be accessed once during the query execution phase, and once when the department.employees collection-valued attribute is used.
D. The database will be accessed once during the query execution phase, once when the Department. Employees collection-valued attribute is used, and once for each employee entity in the Department.employees.

Correct Answer: B
Section: (none)
Explanation

Explanation/Reference:
Explanation:
An important use case for LEFT JOIN is in enabling the prefetching of related data items as a side effect of a query. This is accomplished by specifying the LEFT JOIN as a FETCH JOIN.
10.2.3.5.3. JPQL Fetch Joins

A FETCH JOIN enables the fetching of an association as a side effect of the execution of a query. A FETCH JOIN is specified over an entity and its related entities. The syntax for a fetch join is fetch_join ::= [LEFT [OUTER] | INNER] JOIN FETCH join_association_path_expression The association referenced by the right side of the FETCH JOIN clause must be an association that belongs to an entity that is returned as a result of the query. It is not permitted to specify an identification variable for the entities referenced by the right side of the FETCH JOIN clause, and hence references to the implicitly fetched entities cannot appear elsewhere in the query. The following query returns a set of magazines. As a side effect, the associated articles for those magazines are also retrieved, even though they are not part of the explicit query result. The persistent fields or properties of the articles that are eagerly fetched are fully initialized. The initialization of the relationship properties of the articles that are retrieved is determined by the metadata for the Article entity class.

SELECT mag FROM Magazine mag LEFT JOIN FETCH mag.articles WHERE mag.id = 1

A fetch join has the same join semantics as the corresponding inner or outer join, except that the related objects specified on the right-hand side of the join operation are not returned in the query result or otherwise referenced in the query. Hence, for example, if magazine id 1 has five articles, the above query returns five references to the magazine 1 entity.

References:

QUESTION 42
An application creates a TypedQuery object to perform a query, and sets the query object's flush mode by calling setFlushMode (FlushModeType.commit). The query is executed within a transaction.

Which of the following is true?

A. Updates to the database tables may occur anytime during the transaction associated with the query.
B. Updates to the entities that can affect the outcome of the query cannot be flushed to the database until the transaction commits.
C. Changes to the entities in this transaction cannot be flushed to the database until the transaction commits.
D. setFlushMode cannot be called on a TypedQuery object.

Correct Answer: B
Section: (none)
Explanation

Explanation/Reference:
Explanation:

If FlushModeType.COMMIT is set, the effect of updates made to entities in the persistence context upon queries is unspecified.

Note:
COMMIT:Flushing to occur at transaction commit.
AUTO:(Default) Flushing to occur at query execution.

References:

QUESTION 43
A developer needs to include a set of managed classes in a persistence unit. Which two solutions are correct? (Choose two.)

A. Place the class files in the orm.xml file.
B. Place the class files in the root of the persistence unit.
C. Place the class files in any mapping file that is included on the classpath.
D. Place the class files in any jar on the classpath that is included in the persistence unit.

Correct Answer: BD
Section: (none)
Explanation

Explanation/Reference:
Explanation:
What Persistent Managed Classes Does This Persistence Unit Include?

You can specify the persistent managed classes associated with a persistence unit by using one or more of the following:

*The annotated managed persistence classes contained in the root of the persistence unit.

The root of the persistence unit is the JAR file or directory, whose META-INF directory contains the persistence.xml file. To exclude managed persistence classes, add an <exclude-unlisted-classes> element to the persistence unit.

*<mapping-file> element: specifies one or more object-relational mapping XML files (orm.xml files). *<jar-file> element: specifies one or more JAR files that will be searched for classes.

*<class> element: specifies an explicit list of classes.

References:

QUESTION 44
Entity lifecycle callback methods may be defined in which three classes? (Choose three)

A. Embedded classes
B. Entity classes
C. Abstract classes
D. Entity listener classes
E. Mapped superclasses
F. Concrete non-entity superclasses

Correct Answer: BDE
Section: (none)
Explanation

Explanation/Reference:

Explanation:
Entity Listeners and Callback Methods

A method may be designated as a lifecycle callback method to receive notification of entity lifecycle events. A lifecycle callback method can be defined on an entity class, a mapped superclass, or an entity listener class associated with an entity or mapped superclass.

References:

QUESTION 45
A developer wrote an entity class with the following method:

```
Private static Logger logger = Logger.getLogger ("myLogger");

@PrePersist
@PreUpdate
Public void doA () {
Logger.info ("A");
}
@PostPersist
@PostUpdate
Public void doB () {
logger.info ("B");
}
```

What will the log message contain when an application does the following?

1. Begins a transaction
2. Creates the entity
3. Persists the entity
4. Commits the transaction
5. Begins the entity data
6. Modifies the entity data
7. Merges the entity
8. Commits the second transaction

A. AABB
B. ABAB
C. ABBAB
D. The application will throw an exception because multiple lifecycle callback annotations applied to a single method.

Correct Answer: B
Section: (none)
Explanation

Explanation/Reference:
Explanation:

QUESTION 46
Given the following code:

```
Public void create () {
try {
doA () {
} catch (PersistenceException e) {}
try (doB) ();
} catch (PersistenceException e) {}
}
```

Calling method doA will cause an NonUniqueResultException to be thrown. Calling method doB will cause an EntityExistsException to be thrown.

What two options describe what will happen when the create method is called within an application ' uses container managed transactions? (Choose two)

A. Method doB will never be called.
B. The current transaction will continue after doA executes.
C. The current transaction will continue after doB executes.
D. The current transaction will be marked for rollback when doA is called.
E. The current transaction will be marked for rollback when doB is called.

Correct Answer: BE
Section: (none)
Explanation

Explanation/Reference:
Explanation:
B:
PersistenceExceptionis thrown by the persistence provider when a problem occurs. All instances of PersistenceException except for instances of NoResultException, NonUniqueResultException, LockTimeoutException, and QueryTimeoutException will cause the current transaction, if one is active, to be marked for rollback.

E: EntityExistsException is thrown by the persistence provider when EntityManager.persist(Object) is called and the entity already exists. The current transaction, if one is active, will be marked for rollback.

References:

QUESTION 47
An application that uses pessimistic locking calls an updateData method that results in a LockTimeoutException being thrown. What three statements are correct? (Choose three)

A. The current transaction continues.
B. The current statement continues.
C. The current transaction is rolled back.
D. The current statement is rolled back.
E. The LockTimeoutException can NOT be caught.
F. The LockTimeoutException can be caught, and the updateData method retried.

Correct Answer: ADF
Section: (none)
Explanation

Explanation/Reference:
Explanation:
LockTimeoutExceptionis thrown by the persistence provider when an pessimistic locking conflict occurs that does not result in transaction rollback. This exception may be thrown as part of an API call, at, flush or at commit time. The current transaction, if one is active, will be not be marked for rollback.

References:

QUESTION 48
If a Persistence application locks entity x with a pessimistic lock, which statement is true?

A. The Persistent provider will lock the database row(s) that correspond to all persistent fields of properties of an instance, including element collections.
B. Only single table per class hierarchy mapping is supported with this lock type.
C. A Persistence provider will lock the entity relationships for which the locked entity contains the foreign key.
D. A separate lock statement must be called for each subclass in entity hierarchy.

Correct Answer: C
Section: (none)
Explanation

Explanation/Reference:
Explanation:
NORMAL

public static final PessimisticLockScope NORMAL
This value defines the default behavior for pessimistic locking.

The persistence provider must lock the database row(s) that correspond to the non-collection-valued persistent state of that instance. If a joined inheritance strategy is used, or if the entity is otherwise mapped to a secondary table, this entails locking the row(s) for the entity instance in the additional table(s). Entity relationships for which the locked entity contains the foreign key will also be locked, but not the state of the referenced entities (unless those entities are explicitly locked). Element collections and relationships for which the entity does not contain the foreign key (such as relationships that are mapped to join tables or unidirectional one-to-many relationships for which the target entity contains the foreign key) will not be locked by default.

References:

QUESTION 49
Which one of the following queries selects the customer whose order has the highest total price?

A. CriteriaBuilder cb = ...Criteria Query <Customer> cq = cb.create Query (Customer.class);Root<Customer> c = cq.from(Customer.class);Join<Customer, Order> o =
B. join(Customer__.orders);cq.select(c).distinct(true);Subquery<Double> sq = cq.subquery (Double.class);Root<Order> subo = cq.correlate(o);sq.select(cb.max(subo.get(Order_.totalPrice)));cq.where(cb.equal(o.get(Order_.totalPric e), cb.all(sq)));
C. CriteriaBuilder cb = ...CriteriaQuery<Customer> cq = cb.createquery(customer.class)Root<Customer> c = cq.from(Customer.class);Join<Customer, Order> o = c.join(Customer__.orders);cq.select(c).distinct(true);Subquery<Double> sq = cq.subquery (Double.class);Root<Order> subo = cq.correlate(o);sq.select(cb.max(subo.get(Order_.totalPrice)));cq.where(cb.equal(o.get(Order_.totalPric e), cb.all(sq)));
D. CriteriaBuilder cb = ...CriteriaQuery<Customer> cq = cb.cteateQuery(Customer.class);Root<Customer> c = cq.from(Customer.class);Join<Customer, Order> o = c.join(Customer__.orders);cq.select(c).distinct(true);Subquery<Double> sq = cq.subquery (Double.class);Root<Order> subo = cq.correlate(o);sq.select(cb.max(subo.get(Order_.totalPrice)));cq.where(cb.equal(o.get(Order_.totalPric e), cb.all(sq)));
E. CriteriaBuilder cb = ...CriteriaQuery<Customer> cq = cb.createQuery(Customer.class);Root<Customer> c = cq.from(Customer.class);Join<Customer, Order> o = c.join(Customer_.orders);cq.select(c).distinct(true);Subquery<Double> sq = cq.subquery (Double.class);Root<Order> subo = sq.from(Order.class);sq.select (cb.max (subo.get (Order_ . Total Price))) ;cq.where(sq.all(o.get(Order_.totalPrice)));

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Incorrect: Not a, not b, not C: use .from not .correlate.

Example:
CriteriaBuilder qb = em.getCriteriaBuilder();
CriteriaQuery<Number> cq = qb.createQuery(Number.class); Root<Event> root = cq.from(Event.class);
cq.select(qb.max(root.get("dateProcessed")));
cq.where(qb.equal(Event.get("org"), qb.parameter(MyOrgType.class, "myOrg"))); em.createQuery
(cq).setParameter("myOrg", myOrg).getSingleResult();

Note:
max(Expression<N> x)
Create an aggregate expression applying the numerical max operation.
References:

QUESTION 50
The developer wants to write a criteria query that will return the number of orders made by customer of each county.

Assume that customer is an entity with a unidirectional one-to-many relationship to the Order entity and that Address is an embeddable class, with an attribute country of type String.

Which one of the queries below correctly achieves this?

A. CriteriaBuilder cb> = ...CriteriaQuery cq = cb.createQuery();Root<Customer> c = cq.from (Customer.class);Join<Customer, Order> o =
B. join(Customer_.orders);cq.multiselect(cb.count(0), c,get(customer_.address.get(address_.country)cq.groupBy (c.get(customer_.address) .get (address_.country))
C. CriteriaBuilder cb> = ...CriteriaQuery cq = cb.createQuery();Root<Customer> c = cq.from(Customer.class); cq.select (cb.count(c.join(customer_. Orders)) , c.get(customers(0),
D. get(customer_.address) . get (Address_'country));(c.get(Customer_.address). get(address_.country));
E. CriteriaBuilder cb> = ...CriteriaQuery cq = cb.createQuery();Root<Custower> c = cq.from (Customer.class);Join<Customer, Order> o =
F. join(Customer_.orders);cq.select(cb.count(o));cq.groupBy(c.qet(Customer__.address) - get (Address_.country)) ;
G. CriteriaBuilder cb = ...CriteriaQuery cq = cb.createQueryO;Root<Customer> c = cq.from (Customer.class);Join<Customer, Order> o = c.join(Customer_.orders);Join<Address, String> country = c.join(Customer,.address) .join(Addresscq.multiselect(cq.count(o), country);cq.groupBy(c.get (Customer.address) - get (Address_ . country)) ;

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Incorrect: Not B, Not C: Use multiselect, not select. Not D: Use one Join, not two.

Use multiselect, count and Group By.multiselect() method is used because we are going to get compound result and not one entity type.

Example:
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Object[]> query = cb.createQuery(Object[].class); Root<Department> d = query.from
(Department.class);

Join<Department,Teacher> teachers = d.join("teachers"); query.multiselect(d.get("name"),cb.count (teachers)).groupBy(d.get("name")); References:

QUESTION 51
The developer wants to write a portable criteria query that will order the orders made by customer James Brown according to increasing quantity. Which one of the below queries correctly accomplishes that task?

A. CriteriaBuilder cb= . . .CriteriaQuery<order> cq = cb.createquery<order.class>Root <customer, order>c= cq.from(customer.class);Join <customer, order>o= c.Join(customer_.orders);cq.where (cb.equal(c.get (customer_.name, James Brown)));cq.orderBy (o.get (order_.quantity));
B. CriteriaBuilder cb= . . .CriteriaQuery<order> cq = cb.createquery<order.class>Root <customer, order>c= cq.from(customer.class);Join <customer, order>o= c.Join(customer_.orders);cq.where (cb.equal(c.get (customer_.name, James Brown))); cq.select(o);cq.orderBy (o.get (order_.quantity));
C. CriteriaBuilder cb= . . .CriteriaQuery<order> cq = cb.createquery<order.class>Root <customer, order>c= cq.from(customer.class);Join <customer, order>o= c.Join(customer_.orders);cq.where (cb.equal(c.get (customer_.name, James Brown)));cq.orderBy (o.get (order_.quantity));cq.select(o);
D. CriteriaBuilder cb= . . .CriteriaQuery<order> cq = cb.createquery<order.class>Root <customer, order>c= cq.from(customer.class);Join <customer, order>o= c.Join(customer_.orders);cq.where (cb.equal(c.get (customer_.name, James Brown)));cq.orderBy (o.get (order_.quantity));cq.orderBy ("quantity");

Correct Answer: B
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Incorrect:Not A: Missing select statement. Not C: select statement should not be put last.Not D: Do not use two orderBy.

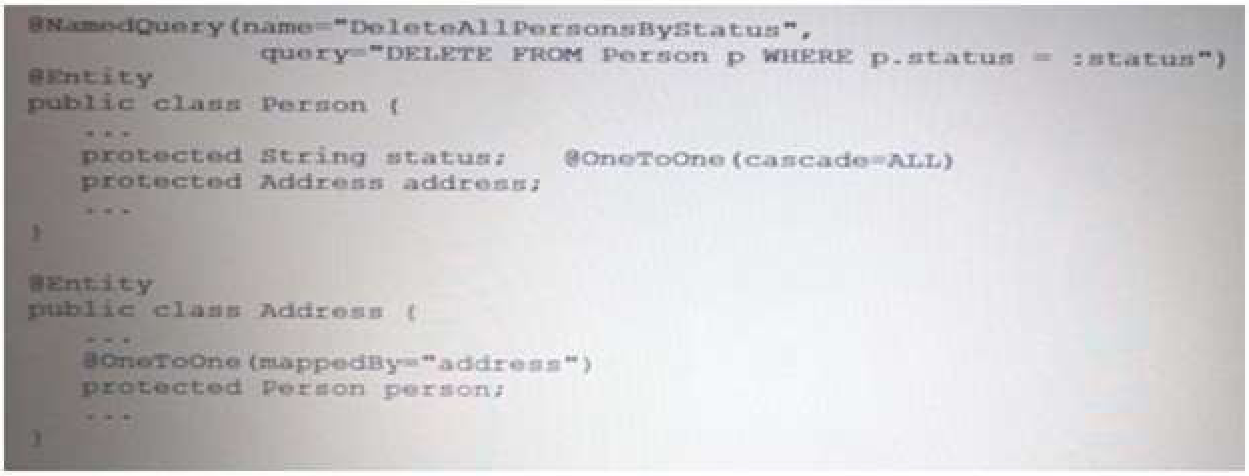
Note:ORDER BY in Criteria Queries
The CriteriaQuery interface provides methods for setting the ORDER BY clause.

For example, the following JPQL query:

SELECT c
FROM Country c
ORDER BY c.currency, c.population DESC
can be built using the criteria query API as follows:

CriteriaQuery<Country> q = cb.createQuery(Country.class); Root<Country> c = q.from(Country.class);
q.select(c);
q.orderBy(cb.asc(c.get("currency")), cb.desc(c.get("population"))); References:

QUESTION 52
An application has two entities, parson and Address.



```
@NamedQuery(name="DeleteAllPersonsByStatus",
            query="DELETE FROM Person p WHERE p.status = :status")
@Entity
public class Person {
    ...
    protected String status;   @OneToOne(cascade=ALL)
    protected Address address;
    ...
}

@Entity
public class Address {
    ...
    @OneToOne(mappedBy="address")
    protected Person person;
    ...
}
```

The application calls the DeletePersonsByStatus named query.

Which of the following is true?

A. The person entities are removed, but NOT their related address entities.
B. The person entities, and all their related address entities, are removed.
C. The DeletePersonsByStatus named query is ill-formed, and will be rejected by the persistence provider.
D. The named query will fall.

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
This is a known issue, hibernate does not perform cascade delete via HQL. If you load the object and delete it using session.delete(object); then the cascade relations are respected. Not via HQL. You have either of the 3 options.

* Load them and call session.delete();
* Load all the child items via HQL, delete them, and then the real objects.
* Put a cascade link at the database level. So when you delete it, the DB will take care of it.

References:

QUESTION 53
An application uses optimistic locking by defining version attributes in its entity classes. The application performs a bulk update of the entities using a JPQL query.

Which of the following is correct?

A. The persistence provider will ensure that the version value in each table is updated.
B. The persistence provider will create a new transaction for the bulk update.
C. An OptimisticLockException will be thrown by the persistence provider.
D. The value of the Version attributes of the updated entitles should be also be explicitly updated by the query.

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
You must take some precautions with bulk update operations. To paraphrase the JPA specification:

bulk updates bypass optimistic locking checks (so you must manually increment the version column and/or manually validate the version column if desired)

Note:The object's version attribute is automatically updated by the JPA provider, and should not normally be modified by the application. The one exception is if the application reads the object in one transaction, sends the object to a client, and updates/merges the object in another transaction. In this case the application must ensure that the original object version is used, otherwise any changes in between the read and write will not be detected.

References:

QUESTION 54
A named query that sets an exclusive pessimistic on the entities returned by the query by setting the NamedQuery lockMode element to LockModeType.PESSIMISTIC_FORCE_INCREMENT. The application starts transaction and executes the query.

Which of the following statements is correct about the entities returned by the query?

A. Only the current transition may modify or delete the entity instances.
B. The current transaction may NOT modify or delete the entity instances.
C. Other concurrent transactions may modify or delete the entity instances.
D. Other concurrent transactions may modify but MAY NOT delete the entity instances.

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
The lock modes LockModeType.PESSIMISTIC_READ, LockModeType.PESSIMISTIC_WRITE, and LockModeType.PESSIMISTIC_FORCE_INCREMENT are used to immediately obtain long-term database locks.

The semantics of requesting locks of type LockModeType.PESSIMISTIC_READ, LockModeType.PESSIMISTIC_WRITE, and LockModeType.PESSIMISTIC_FORCE_INCREMENT are the following.

If transaction T1 calls for a lock of type LockModeType.PESSIMISTIC_READ or LockModeType.PESSIMISTIC_WRITE on an object, the entity manager must ensure that neither of the following phenomena can occur:

P1 (Dirty read): Transaction T1 modifies a row. Another transaction T2 then reads that row and obtains the modified value, before T1 has committed or rolled back. P2 (Non-repeatable read): Transaction T1 reads a row. Another transaction T2 then modifies or deletes that row, before T1 has committed or rolled back.

PESSIMISTIC_FORCE_INCREMENT
Pessimistic write lock, with version update.

References:

QUESTION 55
A developer has created an application managed entity manager.

Which statement is correct?

A. A new persistence context begins when the entity manager is created.
B. A new persistence context begins when a new JTA transaction begins.
C. A new persistence context begins when the entity manager is invoked in the context o\ transaction.
D. A new persistence context begins when the entity manager is invoked in the context of a resource- local transaction.

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Application-Managed Entity Managers
With an application-managed entity managerthe persistence context is not propagated to application components, and the lifecycle of EntityManager instances is managed by the application.

Application-managed entity managers are used when applications need to access a persistence context that is not propagated with the JTA transaction across EntityManager instances in a particular persistence unit. In this case, each EntityManager creates a new, isolated persistence context. The EntityManager and its associated persistence context are created and destroyed explicitly by the application.

References:

QUESTION 56
Given:
```
@PersistenceContext EntityManager em;
public boolean test(Order o) {
boolean b = false;
o = em.merge(o);
em.remove(o);
o = em.merge(o);
b = em.contains(o);
return b;
}
```

Which statement is correct?

A. The method will return TRUE.
B. The method will return FALSE.
C. The method will throw an exception.
D. The order instance will be removed from the database.

Correct Answer: A
Section: (none)
Explanation

Explanation/Reference:
Explanation:
*Order
An object that defines an ordering over the query results.
*merge(T entity)
Merge the state of the given entity into the current persistence context *remove(java.lang.Object entity)
Remove the entity instance.
*contains(java.lang.Object entity)
Check if the instance is a managed entity instance belonging to the current persistence context.

Note:An EntityManager instance is associated with a persistence context. A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance. Within the persistence context, the entity instances and their lifecycle are managed. The EntityManager API is used to create and remove persistent entity instances, to find entities by their primary key, and to query over entities.

The set of entities that can be managed by a given EntityManager instance is defined by a persistence unit. A persistence unit defines the set of all classes that are related or grouped by the application, and which must be colocated in their mapping to a single database

QUESTION 57
If an application uses an extended persistence context, which of the following is true?

A. The persistence context exists until all transactions invoked by the EntityManager complete.
B. The persistence context exists until all transactions invoked by the EntityManagar complete and the EntityManager.clear () method is invoked.
C. The persistence context exists until the EntityManager instance is closed.
D. The persistence context exists until the EntityManagerFactory instance is closed.

Correct Answer: C
Section: (none)
Explanation

Explanation/Reference:
Explanation:
The extended persistence context exists from the point at which the entity manager has been created using EntityManagerFactory.createEntityManager until the entity manager is closed by means of EntityManager.close.

References:

QUESTION 58
An application uses an application-managed entity manager. Which of the following is NOT true?

A. The application may specify whether the scope of the persistence context is extended.
B. The application must use EntityManagerFactory instances to create entity managers.
C. Entity manager instances must be explicitly closed.
D. The application may need to call EntityManager. joinTransactionif a JTA aware entity manager is used.

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
When a JTA application-managed entity manager is used, if the entity manager is created outside the scope of the current JTA transaction, it is the responsibility of the application to associate the entity manager with the transaction (if desired) by calling EntityManager.joinTransaction.

References:

QUESTION 59
An application that uses container-managed transaction demarcation creates a query within an active transaction and receives a QueryTimeoutException. Which of those scenarios describes what happens to the active transaction?

A. The statement and the transaction continue.
B. The query is recreated within the current transaction.
C. The statement and the transaction are marked for rollback.
D. The statement is rolled back, but the transaction continues.

Correct Answer: D
Section: (none)
Explanation

Explanation/Reference:
Explanation:
QueryTimeoutExceptionis thrown by the persistence provider when a query times out and only the statement is rolled back. The current transaction, if one is active, will be not be marked for rollback.

References:

QUESTION 60
The developer has defined the following entity class office:

```
@Entity
public class Office {
@Id
private int id;
private String name;
@OneToMany
private List<Room> rooms;
}
```

Which of the following attributes will be in corresponding generated static metamodel class for the rooms' field?

A. Public static volatile CollectionAttribute<Room> rooms;
B. Public static volatile ListAttribute<Room> rooms;
C. Public static volatile ListAttribute<Office, Room> rooms;
D. Public static volatile SingleAttribute<Room> rooms;

Correct Answer: C
Section: (none)
Explanation

Explanation/Reference:
Explanation:
*Annotation Type OneToMany
Defines a many-valued association with one-to-many multiplicity.
*Interface ListAttribute<X,E>

Type Parameters:
X - The type the represented List belongs to
E - The element type of the represented List

Incorrect:
not A: A collection cannot reprecent a one to many relationship.

References:

QUESTION 61
Given two entities with many-to-many bidirectional association between them:

```
@Entity public class Employee {
Collection projects;
// more code here
}
```

and

```
@Entity public class Project {
Set<Employee> emps;
// more code here
}
```

What set of annotations correctly defines the association?

A. @manyToMany on the projects field,@manyToMany (mappedBy= "projects") on the emps field
B. @manyToMany (mappedBy = emps) on the projects field,@manyToMany on the emps field
C. @manyToMany (targetEntity = project.class) on the projects field,@manyToMany (mappedBy = "projects") on the emps field
D. @manyToMany (targetEntity =project.class) on the projects field,@manyToMany on the emps field

Correct Answer: C
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Every many to many relationship has obviously two sides: a Owning side and the Inverse side (which is... non-owning).
Here the Employee class is the owning side.

targetEntity:The entity class that is the target of the association.The dot (".") notation syntax must be used in the mappedBy element to indicate the relationship attribute within the embedded attribute.

This mappedBy element is mandatory and must be placed on the Inverse side of the relationship in case we are mapping a Bidirectional.Here Projects class is on the inverse side.

Note: @manyToMany defines a many-valued association with many-to-many multiplicity.

Every many-to-many association has two sides, the owning side and the non-owning, or inverse, side. The join table is specified on the owning side. If the association is bidirectional, either side may be designated as the owning side. If the relationship is bidirectional, the non-owning side must use the mappedBy element of the ManyToMany annotation to specify the relationship field or property of the owning side.

References:

QUESTION 62
An application wants to utilize side effects of cascading entity manager operations to related entities.

Which statement is correct?

A. The persist operation is always cascaded to related entitles for one-to one and one-to-many relationships.
B. To minimize the effect of the remove operation applied to an entity participating in a many-to-many relationship the remove operation should he cascade to entities on both sides of the relationship.
C. The persist operation applied to a new entity x is cascaded to entities referenced by x if the relationship from x to these other entities is annotated with the cascade=PERSIST or cascade=ALL annotation element value.
D. The remove operation applied to a removed entity x is cascaded to entities referenced by x of the relationship from x to these other entities is annotated with the cascade = REMOVE of cascade = ALL annotation element value.

Correct Answer: C
Section: (none)
Explanation

Explanation/Reference:

Explanation:
The semantics of the flush operation, applied to an entity X are as follows:

*If X is a managed entity, it is synchronized to the database. /For all entities Y referenced by a relationship from X, if the relationship to Y has been annotated with the cascade element value cascade=PERSIST or cascade=ALL, the persist operation is applied to Y.

References:

QUESTION 63
Given two entities with one to-one association:

```
@Entity public class Person {
@Id Integer id;
...
}

@Entity public class PersonDetail {
@Id Integer id;
...
}
```

Which code fragment correctly defines the detail field that PersonDetail instance in removed if the person instance that references it is removed?

A. @OneToOne (optional = false)personDetail detail;
B. @OneToOne (optional = false)@mapsIdPersonDetail Detail;
C. @ OneToOne (orphanremoval = true)PersonDetail Detail;
D. @ OneToOne (cascade = ORPHAN _ DELETE)@mapsIdPersonDetail detail;

Correct Answer: C
Section: (none)
Explanation

Explanation/Reference:
Explanation:
Orphan Removal in Relationships
When a target entity in one-to-one or one-to-many relationship is removed from the relationship, it is often desirable to cascade the remove operation to the target entity. Such target entities are considered "orphans," and the orphanRemoval attribute can be used to specify that orphaned entities should be removed. For example, if an order has many line items and one of them is removed from the order, the removed line item is considered an orphan. If orphanRemoval is set to true, the line item entity will be deleted when the line item is removed from the order.

The orphanRemoval attribute in @OneToMany and @oneToOne takes a Boolean value and is by default false.

The following example will cascade the remove operation to the orphaned customer entity when it is removed from the relationship:

```
@OneToMany(mappedBy="customer", orphanRemoval="true")
public List<Order> getOrders() { ... }
```

References: