javax.persistence

Annotation Type PersistenceContext

```
@Target(value={TYPE,METHOD,FIELD})
 @Retention(value=RUNTIME)
public @interface PersistenceContext
```

Expresses a dependency on a container-managed `EntityManager` and its associated persistence context.

Since:

`Java Persistence 1.0`

Optional Element Summary

Optional Elements

Modifier and Type	Optional Element and Description
`String`	**`name`** (Optional) The name by which the entity manager is to be accessed in the environment referencing context; not needed when dependency injection is used.
`PersistenceProperty[]`	**`properties`** (Optional) Properties for the container or persistence provider.
`SynchronizationType`	**`synchronization`** (Optional) Specifies whether the persistence context is always automatically synchronized with the current transaction or whether the persistence context must be explicitly joined to the current transaction by means of the EntityManager `joinTransaction` method.
`PersistenceContextType`	**`type`** (Optional) Specifies whether a transaction-scoped persistence context or an extended persistence context is to be used.
`String`	**`unitName`** (Optional) The name of the persistence unit as defined in the `persistence.xml` file.

Element Detail

name

```
public abstract String name
```

(Optional) The name by which the entity manager is to be accessed in the environment referencing context; not needed when dependency injection is used.

Default:

""

unitName

```
public abstract String unitName
```

(Optional) The name of the persistence unit as defined in the `persistence.xml` file. If the `unitName` element is specified, the persistence unit for the entity manager that is accessible in JNDI must have the same name.

Default:

""

type

```
public abstract PersistenceContextType type
```

(Optional) Specifies whether a transaction-scoped persistence context or an extended persistence context is to be used.

Default:

`javax.persistence.PersistenceContextType.TRANSACTION`

synchronization

```
public abstract SynchronizationType synchronization
```

(Optional) Specifies whether the persistence context is always automatically synchronized with the current transaction or whether the persistence context must be explicitly joined to the current transaction by means of the EntityManager `joinTransaction` method.

Since:

`Java Persistence 2.1`

Default:

`javax.persistence.SynchronizationType.SYNCHRONIZED`

properties

```
public abstract PersistenceProperty[] properties
```

(Optional) Properties for the container or persistence provider. Vendor specific properties may be included in this set of properties. Properties that are not recognized by a vendor are ignored.

Default:

{}

javax.persistence

Interface EntityManager

```
public interface EntityManager
```

Interface used to interact with the persistence context.

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.

Since:

`Java Persistence 1.0`

See Also:

`Query`, `TypedQuery`, `CriteriaQuery`, `PersistenceContext`, `StoredProcedureQuery`

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
`void`	**`clear`**`()` Clear the persistence context, causing all managed entities to become detached.
`void`	**`close`**`()` Close an application-managed entity manager.
`boolean`	**`contains`**`(`**`Object`** `entity)` Check if the instance is a managed entity instance belonging to the current persistence context.
`<T>` **`EntityGraph`**`<T>`	**`createEntityGraph`**`(`**`Class`**`<T> rootType)` Return a mutable EntityGraph that can be used to dynamically create an EntityGraph.
`EntityGraph``<?>`	**`createEntityGraph`**`(`**`String`** `graphName)` Return a mutable copy of the named EntityGraph.
`Query`	**`createNamedQuery`**`(`**`String`** `name)`

	Create an instance of Query for executing a named query (in the Java Persistence query language or in native SQL).
<T> TypedQuery<T>	createNamedQuery(String name, Class<T> resultClass) Create an instance of TypedQuery for executing a Java Persistence query language named query.
StoredProcedureQuery	createNamedStoredProcedureQuery(String name) Create an instance of StoredProcedureQuery for executing a stored procedure in the database.
Query	createNativeQuery(String sqlString) Create an instance of Query for executing a native SQL statement, e.g., for update or delete.
Query	createNativeQuery(String sqlString, Class resultClass) Create an instance of Query for executing a native SQL query.
Query	createNativeQuery(String sqlString, String resultSetMapping) Create an instance of Query for executing a native SQL query.
Query	createQuery(CriteriaDelete deleteQuery) Create an instance of Query for executing a criteria delete query.
<T> TypedQuery<T>	createQuery(CriteriaQuery<T> criteriaQuery) Create an instance of TypedQuery for executing a criteria query.
Query	createQuery(CriteriaUpdate updateQuery) Create an instance of Query for executing a criteria update query.
Query	createQuery(String qlString) Create an instance of Query for executing a Java Persistence query language statement.
<T> TypedQuery<T>	createQuery(String qlString, Class<T> resultClass) Create an instance of TypedQuery for executing a Java Persistence query language statement.
StoredProcedureQuery	createStoredProcedureQuery(String procedureName) Create an instance of StoredProcedureQuery for executing a stored procedure in the database.
StoredProcedureQuery	createStoredProcedureQuery(String procedureName, Class... resultClasses) Create an instance of StoredProcedureQuery for executing a stored procedure in the database.
StoredProcedureQuery	createStoredProcedureQuery(String procedureName,

	String... resultSetMappings) Create an instance of StoredProcedureQuery for executing a stored procedure in the database.
void	**detach**(**Object** entity) Remove the given entity from the persistence context, causing a managed entity to become detached.
<T> T	**find**(**Class**<T> entityClass, **Object** primaryKey) Find by primary key.
<T> T	**find**(**Class**<T> entityClass, **Object** primaryKey, **LockModeType** lockMode) Find by primary key and lock.
<T> T	**find**(**Class**<T> entityClass, **Object** primaryKey, **LockModeType** lockMode, **Map**<**String**,**Object**> properties) Find by primary key and lock, using the specified properties.
<T> T	**find**(**Class**<T> entityClass, **Object** primaryKey, **Map**<**String**,**Object**> properties) Find by primary key, using the specified properties.
void	**flush**() Synchronize the persistence context to the underlying database.
CriteriaBuilder	**getCriteriaBuilder**() Return an instance of CriteriaBuilder for the creation of CriteriaQuery objects.
Object	**getDelegate**() Return the underlying provider object for the EntityManager, if available.
EntityGraph<?>	**getEntityGraph**(**String** graphName) Return a named EntityGraph.
<T> **List**<**EntityGraph**<? super T>>	**getEntityGraphs**(**Class**<T> entityClass) Return all named EntityGraphs that have been defined for the provided class type.
EntityManagerFactory	**getEntityManagerFactory**() Return the entity manager factory for the entity manager.
FlushModeType	**getFlushMode**() Get the flush mode that applies to all objects contained in the persistence context.
LockModeType	**getLockMode**(**Object** entity) Get the current lock mode for the entity instance.
Metamodel	**getMetamodel**() Return an instance of Metamodel interface for access to the metamodel of the persistence unit.

Map<String,Object>	**getProperties**() Get the properties and hints and associated values that are in effect for the entity manager.
<T> T	**getReference**(Class<T> entityClass, Object primaryKey) Get an instance, whose state may be lazily fetched.
EntityTransaction	**getTransaction**() Return the resource-level EntityTransaction object.
boolean	**isJoinedToTransaction**() Determine whether the entity manager is joined to the current transaction.
boolean	**isOpen**() Determine whether the entity manager is open.
void	**joinTransaction**() Indicate to the entity manager that a JTA transaction is active and join the persistence context to it.
void	**lock**(Object entity, LockModeType lockMode) Lock an entity instance that is contained in the persistence context with the specified lock mode type.
void	**lock**(Object entity, LockModeType lockMode, Map<String,Object> properties) Lock an entity instance that is contained in the persistence context with the specified lock mode type and with specified properties.
<T> T	**merge**(T entity) Merge the state of the given entity into the current persistence context.
void	**persist**(Object entity) Make an instance managed and persistent.
void	**refresh**(Object entity) Refresh the state of the instance from the database, overwriting changes made to the entity, if any.
void	**refresh**(Object entity, LockModeType lockMode) Refresh the state of the instance from the database, overwriting changes made to the entity, if any, and lock it with respect to given lock mode type.
void	**refresh**(Object entity, LockModeType lockMode, Map<String,Object> properties) Refresh the state of the instance from the database, overwriting changes made to the entity, if any, and lock it with respect to given lock mode type and with specified properties.
void	**refresh**(Object entity,

	`Map<String,Object> properties)` Refresh the state of the instance from the database, using the specified properties, and overwriting changes made to the entity, if any.
`void`	`remove(Object entity)` Remove the entity instance.
`void`	`setFlushMode(FlushModeType flushMode)` Set the flush mode that applies to all objects contained in the persistence context.
`void`	`setProperty(String propertyName, Object value)` Set an entity manager property or hint.
`<T> T`	`unwrap(Class<T> cls)` Return an object of the specified type to allow access to the provider-specific API.

Method Detail

persist

```
void persist(Object entity)
```

Make an instance managed and persistent.

Parameters:

`entity - entity instance`

Throws:

`EntityExistsException - if the entity already exists. (If the entity already exists, the EntityExistsException may be thrown when the persist operation is invoked, or the EntityExistsException or another PersistenceException may be thrown at flush or commit time.)`

`IllegalArgumentException - if the instance is not an entity`

`TransactionRequiredException - if there is no transaction when invoked on a container-managed entity manager of that is of type PersistenceContextType.TRANSACTION`

merge

```
<T> T merge(T entity)
```

Merge the state of the given entity into the current persistence context.

Parameters:

`entity - entity instance`

Returns:

`the managed instance that the state was merged to`

Throws:

IllegalArgumentException - if instance is not an entity or is a removed entity

TransactionRequiredException - if there is no transaction when invoked on a container-managed entity manager of that is of type PersistenceContextType.TRANSACTION

remove

```
void remove(Object entity)
```

Remove the entity instance.

Parameters:

entity - entity instance

Throws:

IllegalArgumentException - if the instance is not an entity or is a detached entity

TransactionRequiredException - if invoked on a container-managed entity manager of type PersistenceContextType.TRANSACTION and there is no transaction

find

```
<T> T find(Class<T> entityClass,
           Object primaryKey)
```

Find by primary key. Search for an entity of the specified class and primary key. If the entity instance is contained in the persistence context, it is returned from there.

Parameters:

entityClass - entity class

primaryKey - primary key

Returns:

the found entity instance or null if the entity does not exist

Throws:

IllegalArgumentException - if the first argument does not denote an entity type or the second argument is is not a valid type for that entity's primary key or is null

find

```
<T> T find(Class<T> entityClass,
           Object primaryKey,
           Map<String,Object> properties)
```

Find by primary key, using the specified properties. Search for an entity of the specified class and primary key. If the entity instance is contained in the persistence context, it is returned from there. If a vendor-specific property or hint is not recognized, it is silently ignored.

Parameters:

entityClass - entity class

primaryKey - primary key

properties - standard and vendor-specific properties and hints

Returns:

the found entity instance or null if the entity does not exist

Throws:

IllegalArgumentException - if the first argument does not denote an entity type or the second argument is is not a valid type for that entity's primary key or is null

Since:

Java Persistence 2.0

find

```
<T> T find(Class<T> entityClass,
           Object primaryKey,
           LockModeType lockMode)
```

Find by primary key and lock. Search for an entity of the specified class and primary key and lock it with respect to the specified lock type. If the entity instance is contained in the persistence context, it is returned from there, and the effect of this method is the same as if the lock method had been called on the entity.

If the entity is found within the persistence context and the lock mode type is pessimistic and the entity has a version attribute, the persistence provider must perform optimistic version checks when obtaining the database lock. If these checks fail, the `OptimisticLockException` will be thrown.

If the lock mode type is pessimistic and the entity instance is found but cannot be locked:

- the `PessimisticLockException` will be thrown if the database locking failure causes transaction-level rollback
- the `LockTimeoutException` will be thrown if the database locking failure causes only statement-level rollback

Parameters:

entityClass - entity class

primaryKey - primary key

lockMode - lock mode

Returns:

the found entity instance or null if the entity does not exist

Throws:

IllegalArgumentException - if the first argument does not denote an entity type or the second argument is not a valid type for that entity's primary key or is null

TransactionRequiredException - if there is no transaction and a lock mode other than NONE is specified or if invoked on an entity manager which has not been joined to the current transaction and a lock mode other than NONE is specified

OptimisticLockException - if the optimistic version check fails

PessimisticLockException - if pessimistic locking fails and the transaction is rolled back

LockTimeoutException - if pessimistic locking fails and only the statement is rolled back

PersistenceException - if an unsupported lock call is made

Since:

Java Persistence 2.0

find

```
<T> T find(Class<T> entityClass,
           Object primaryKey,
           LockModeType lockMode,
           Map<String,Object> properties)
```

Find by primary key and lock, using the specified properties. Search for an entity of the specified class and primary key and lock it with respect to the specified lock type. If the entity instance is contained in the persistence context, it is returned from there.

If the entity is found within the persistence context and the lock mode type is pessimistic and the entity has a version attribute, the persistence provider must perform optimistic version checks when obtaining the database lock. If these checks fail, the `OptimisticLockException` will be thrown.

If the lock mode type is pessimistic and the entity instance is found but cannot be locked:

- the `PessimisticLockException` will be thrown if the database locking failure causes transaction-level rollback
- the `LockTimeoutException` will be thrown if the database locking failure causes only statement-level rollback

If a vendor-specific property or hint is not recognized, it is silently ignored.

Portable applications should not rely on the standard timeout hint. Depending on the database in use and the locking mechanisms used by the provider, the hint may or may not be observed.

Parameters:

entityClass - entity class

primaryKey - primary key

lockMode - lock mode

properties - standard and vendor-specific properties and hints

Returns:

the found entity instance or null if the entity does not exist

Throws:

IllegalArgumentException - if the first argument does not denote an entity type or the second argument is not a valid type for that entity's primary key or is null

TransactionRequiredException - if there is no transaction and a lock mode other than NONE is specified or if invoked on an entity manager which has not been joined to the current transaction and a lock mode other than NONE is specified

OptimisticLockException - if the optimistic version check fails

PessimisticLockException - if pessimistic locking fails and the transaction is rolled back

LockTimeoutException - if pessimistic locking fails and only the statement is rolled back

PersistenceException - if an unsupported lock call is made

Since:

Java Persistence 2.0

getReference

```
<T> T getReference(Class<T> entityClass,
                   Object primaryKey)
```

Get an instance, whose state may be lazily fetched. If the requested instance does not exist in the database, the EntityNotFoundException is thrown when the instance state is first accessed. (The persistence provider runtime is permitted to throw the EntityNotFoundException when getReference is called.) The application should not expect that the instance state will be available upon detachment, unless it was accessed by the application while the entity manager was open.

Parameters:

entityClass - entity class

primaryKey - primary key

Returns:

the found entity instance

Throws:

IllegalArgumentException - if the first argument does not denote an entity type or the second argument is not a valid type for that entity's primary key or is null

EntityNotFoundException - if the entity state cannot be accessed

flush

```
void flush()
```

Synchronize the persistence context to the underlying database.

Throws:

TransactionRequiredException - if there is no transaction or if the entity manager has not been joined to the current transaction

PersistenceException - if the flush fails

setFlushMode

```
void setFlushMode(FlushModeType flushMode)
```

Set the flush mode that applies to all objects contained in the persistence context.

Parameters:

`flushMode - flush mode`

getFlushMode

```
FlushModeType getFlushMode()
```

Get the flush mode that applies to all objects contained in the persistence context.

Returns:

`flushMode`

lock

```
void lock(Object entity,
          LockModeType lockMode)
```

Lock an entity instance that is contained in the persistence context with the specified lock mode type.

If a pessimistic lock mode type is specified and the entity contains a version attribute, the persistence provider must also perform optimistic version checks when obtaining the database lock. If these checks fail, the `OptimisticLockException` will be thrown.

If the lock mode type is pessimistic and the entity instance is found but cannot be locked:

- the `PessimisticLockException` will be thrown if the database locking failure causes transaction-level rollback
- the `LockTimeoutException` will be thrown if the database locking failure causes only statement-level rollback

Parameters:

`entity - entity instance`

`lockMode - lock mode`

Throws:

`IllegalArgumentException - if the instance is not an entity or is a detached entity`

`TransactionRequiredException - if there is no transaction or if invoked on an entity manager which has not been joined to the current transaction`

`EntityNotFoundException - if the entity does not exist in the database when pessimistic locking is performed`

`OptimisticLockException - if the optimistic version check fails`

`PessimisticLockException - if pessimistic locking fails and the transaction is rolled back`

`LockTimeoutException - if pessimistic locking fails and only the statement is rolled back`

`PersistenceException - if an unsupported lock call is made`

lock

```
void lock(Object entity,
          LockModeType lockMode,
          Map<String,Object> properties)
```

Lock an entity instance that is contained in the persistence context with the specified lock mode type and with specified properties.

If a pessimistic lock mode type is specified and the entity contains a version attribute, the persistence provider must also perform optimistic version checks when obtaining the database lock. If these checks fail, the `OptimisticLockException` will be thrown.

If the lock mode type is pessimistic and the entity instance is found but cannot be locked:

- the `PessimisticLockException` will be thrown if the database locking failure causes transaction-level rollback
- the `LockTimeoutException` will be thrown if the database locking failure causes only statement-level rollback

If a vendor-specific property or hint is not recognized, it is silently ignored.

Portable applications should not rely on the standard timeout hint. Depending on the database in use and the locking mechanisms used by the provider, the hint may or may not be observed.

Parameters:

`entity` - entity instance

`lockMode` - lock mode

`properties` - standard and vendor-specific properties and hints

Throws:

`IllegalArgumentException` - if the instance is not an entity or is a detached entity

`TransactionRequiredException` - if there is no transaction or if invoked on an entity manager which has not been joined to the current transaction

`EntityNotFoundException` - if the entity does not exist in the database when pessimistic locking is performed

`OptimisticLockException` - if the optimistic version check fails

`PessimisticLockException` - if pessimistic locking fails and the transaction is rolled back

`LockTimeoutException` - if pessimistic locking fails and only the statement is rolled back

`PersistenceException` - if an unsupported lock call is made

Since:

`Java Persistence 2.0`

refresh

```
void refresh(Object entity)
```

Refresh the state of the instance from the database, overwriting changes made to the entity, if any.

Parameters:

entity - entity instance

Throws:

IllegalArgumentException - if the instance is not an entity or the entity is not managed

TransactionRequiredException - if there is no transaction when invoked on a container-managed entity manager of type PersistenceContextType.TRANSACTION

EntityNotFoundException - if the entity no longer exists in the database

refresh

```
void refresh(Object entity,
             Map<String,Object> properties)
```

Refresh the state of the instance from the database, using the specified properties, and overwriting changes made to the entity, if any.

If a vendor-specific property or hint is not recognized, it is silently ignored.

Parameters:

entity - entity instance

properties - standard and vendor-specific properties and hints

Throws:

IllegalArgumentException - if the instance is not an entity or the entity is not managed

TransactionRequiredException - if there is no transaction when invoked on a container-managed entity manager of type PersistenceContextType.TRANSACTION

EntityNotFoundException - if the entity no longer exists in the database

Since:

Java Persistence 2.0

refresh

```
void refresh(Object entity,
             LockModeType lockMode)
```

Refresh the state of the instance from the database, overwriting changes made to the entity, if any, and lock it with respect to given lock mode type.

If the lock mode type is pessimistic and the entity instance is found but cannot be locked:

- the `PessimisticLockException` will be thrown if the database locking failure causes transaction-level rollback
- the `LockTimeoutException` will be thrown if the database locking failure causes only statement-level rollback.

Parameters:

entity - entity instance

lockMode - lock mode

Throws:

`IllegalArgumentException` - if the instance is not an entity or the entity is not managed

`TransactionRequiredException` - if invoked on a container-managed entity manager of type PersistenceContextType.TRANSACTION when there is no transaction; if invoked on an extended entity manager when there is no transaction and a lock mode other than NONE has been specified; or if invoked on an extended entity manager that has not been joined to the current transaction and a lock mode other than NONE has been specified

`EntityNotFoundException` - if the entity no longer exists in the database

`PessimisticLockException` - if pessimistic locking fails and the transaction is rolled back

`LockTimeoutException` - if pessimistic locking fails and only the statement is rolled back

`PersistenceException` - if an unsupported lock call is made

Since:

Java Persistence 2.0

refresh

```
void refresh(Object entity,
             LockModeType lockMode,
             Map<String,Object> properties)
```

Refresh the state of the instance from the database, overwriting changes made to the entity, if any, and lock it with respect to given lock mode type and with specified properties.

If the lock mode type is pessimistic and the entity instance is found but cannot be locked:

- the `PessimisticLockException` will be thrown if the database locking failure causes transaction-level rollback
- the `LockTimeoutException` will be thrown if the database locking failure causes only statement-level rollback

If a vendor-specific property or hint is not recognized, it is silently ignored.

Portable applications should not rely on the standard timeout hint. Depending on the database in use and the locking mechanisms used by the provider, the hint may or may not be observed.

Parameters:

`entity` - entity instance

`lockMode` - lock mode

`properties` - standard and vendor-specific properties and hints

Throws:

`IllegalArgumentException` - if the instance is not an entity or the entity is not managed

`TransactionRequiredException` - if invoked on a container-managed entity manager of type PersistenceContextType.TRANSACTION when there is no transaction; if invoked on an extended entity manager when there is no transaction and a lock mode other

than NONE has been specified; or if invoked on an extended entity manager that has not been joined to the current transaction and a lock mode other than NONE has been specified

EntityNotFoundException - if the entity no longer exists in the database

PessimisticLockException - if pessimistic locking fails and the transaction is rolled back

LockTimeoutException - if pessimistic locking fails and only the statement is rolled back

PersistenceException - if an unsupported lock call is made

Since:

Java Persistence 2.0

clear

```
void clear()
```

Clear the persistence context, causing all managed entities to become detached. Changes made to entities that have not been flushed to the database will not be persisted.

detach

```
void detach(Object entity)
```

Remove the given entity from the persistence context, causing a managed entity to become detached. Unflushed changes made to the entity if any (including removal of the entity), will not be synchronized to the database. Entities which previously referenced the detached entity will continue to reference it.

Parameters:

entity - entity instance

Throws:

IllegalArgumentException - if the instance is not an entity

Since:

Java Persistence 2.0

contains

```
boolean contains(Object entity)
```

Check if the instance is a managed entity instance belonging to the current persistence context.

Parameters:

entity - entity instance

Returns:

boolean indicating if entity is in persistence context

Throws:

IllegalArgumentException - if not an entity

getLockMode

```
LockModeType getLockMode(Object entity)
```

Get the current lock mode for the entity instance.

Parameters:

`entity - entity instance`

Returns:

`lock mode`

Throws:

`TransactionRequiredException - if there is no transaction or if the entity manager has not been joined to the current transaction`

`IllegalArgumentException - if the instance is not a managed entity and a transaction is active`

Since:

`Java Persistence 2.0`

setProperty

```
void setProperty(String propertyName,
                 Object value)
```

Set an entity manager property or hint. If a vendor-specific property or hint is not recognized, it is silently ignored.

Parameters:

`propertyName - name of property or hint`

`value - value for property or hint`

Throws:

`IllegalArgumentException - if the second argument is not valid for the implementation`

Since:

`Java Persistence 2.0`

getProperties

```
Map<String,Object> getProperties()
```

Get the properties and hints and associated values that are in effect for the entity manager. Changing the contents of the map does not change the configuration in effect.

Returns:

`map of properties and hints in effect for entity manager`

Since:

`Java Persistence 2.0`

createQuery

```
Query createQuery(String qlString)
```

Create an instance of `Query` for executing a Java Persistence query language statement.

Parameters:

`qlString - a Java Persistence query string`

Returns:

`the new query instance`

Throws:

`IllegalArgumentException - if the query string is found to be invalid`

createQuery

```
<T> TypedQuery<T> createQuery(CriteriaQuery<T> criteriaQuery)
```

Create an instance of `TypedQuery` for executing a criteria query.

Parameters:

`criteriaQuery - a criteria query object`

Returns:

`the new query instance`

Throws:

`IllegalArgumentException - if the criteria query is found to be invalid`

Since:

`Java Persistence 2.0`

createQuery

```
Query createQuery(CriteriaUpdate updateQuery)
```

Create an instance of `Query` for executing a criteria update query.

Parameters:

`updateQuery - a criteria update query object`

Returns:

`the new query instance`

Throws:

`IllegalArgumentException - if the update query is found to be invalid`

Since:

`Java Persistence 2.1`

createQuery

```
Query createQuery(CriteriaDelete deleteQuery)
```

Create an instance of `Query` for executing a criteria delete query.

Parameters:

`deleteQuery - a criteria delete query object`

Returns:

`the new query instance`

Throws:

`IllegalArgumentException - if the delete query is found to be invalid`

Since:

`Java Persistence 2.1`

createQuery

```
<T> TypedQuery<T> createQuery(String qlString,
                              Class<T> resultClass)
```

Create an instance of `TypedQuery` for executing a Java Persistence query language statement. The select list of the query must contain only a single item, which must be assignable to the type specified by the `resultClass` argument.

Parameters:

`qlString - a Java Persistence query string`

`resultClass - the type of the query result`

Returns:

`the new query instance`

Throws:

`IllegalArgumentException - if the query string is found to be invalid or if the query result is found to not be assignable to the specified type`

Since:

`Java Persistence 2.0`

createNamedQuery

```
Query createNamedQuery(String name)
```

Create an instance of `Query` for executing a named query (in the Java Persistence query language or in native SQL).

Parameters:

`name - the name of a query defined in metadata`

Returns:

`the new query instance`

Throws:

`IllegalArgumentException - if a query has not been defined with the given name or if the query string is found to be invalid`

createNamedQuery

```
<T> TypedQuery<T> createNamedQuery(String name,
                                   Class<T> resultClass)
```

Create an instance of `TypedQuery` for executing a Java Persistence query language named query. The select list of the query must contain only a single item, which must be assignable to the type specified by the `resultClass` argument.

Parameters:

`name - the name of a query defined in metadata`

`resultClass - the type of the query result`

Returns:

`the new query instance`

Throws:

`IllegalArgumentException - if a query has not been defined with the given name or if the query string is found to be invalid or if the query result is found to not be assignable to the specified type`

Since:

`Java Persistence 2.0`

createNativeQuery

```
Query createNativeQuery(String sqlString)
```

Create an instance of `Query` for executing a native SQL statement, e.g., for update or delete. If the query is not an update or delete query, query execution will result in each row of the SQL result being returned as a result of type Object[] (or a result of type Object if there is only one column in the select list.) Column values are returned in the order of their appearance in the select list and default JDBC type mappings are applied.

Parameters:

`sqlString - a native SQL query string`

Returns:

`the new query instance`

createNativeQuery

```
Query createNativeQuery(String sqlString,
                        Class resultClass)
```

Create an instance of `Query` for executing a native SQL query.

Parameters:

`sqlString - a native SQL query string`

`resultClass - the class of the resulting instance(s)`

Returns:

`the new query instance`

createNativeQuery

```
Query createNativeQuery(String sqlString,
                        String resultSetMapping)
```

Create an instance of Query for executing a native SQL query.

Parameters:

sqlString - a native SQL query string

resultSetMapping - the name of the result set mapping

Returns:

the new query instance

createNamedStoredProcedureQuery

```
StoredProcedureQuery createNamedStoredProcedureQuery(String name)
```

Create an instance of StoredProcedureQuery for executing a stored procedure in the database.

Parameters must be registered before the stored procedure can be executed.

If the stored procedure returns one or more result sets, any result set will be returned as a list of type Object[].

Parameters:

name - name assigned to the stored procedure query in metadata

Returns:

the new stored procedure query instance

Throws:

IllegalArgumentException - if a query has not been defined with the given name

Since:

Java Persistence 2.1

createStoredProcedureQuery

```
StoredProcedureQuery createStoredProcedureQuery(String procedureName)
```

Create an instance of StoredProcedureQuery for executing a stored procedure in the database.

Parameters must be registered before the stored procedure can be executed.

If the stored procedure returns one or more result sets, any result set will be returned as a list of type Object[].

Parameters:

procedureName - name of the stored procedure in the database

Returns:

the new stored procedure query instance

Throws:

IllegalArgumentException - if a stored procedure of the given name does not exist (or the query execution will fail)

Since:

Java Persistence 2.1

createStoredProcedureQuery

```
StoredProcedureQuery createStoredProcedureQuery(String procedureName,
                                                Class... resultClasses)
```

Create an instance of StoredProcedureQuery for executing a stored procedure in the database.

Parameters must be registered before the stored procedure can be executed.

The resultClass arguments must be specified in the order in which the result sets will be returned by the stored procedure invocation.

Parameters:

procedureName - name of the stored procedure in the database

resultClasses - classes to which the result sets produced by the stored procedure are to be mapped

Returns:

the new stored procedure query instance

Throws:

IllegalArgumentException - if a stored procedure of the given name does not exist (or the query execution will fail)

Since:

Java Persistence 2.1

createStoredProcedureQuery

```
StoredProcedureQuery createStoredProcedureQuery(String procedureName,
                                                String... resultSetMappings)
```

Create an instance of StoredProcedureQuery for executing a stored procedure in the database.

Parameters must be registered before the stored procedure can be executed.

The resultSetMapping arguments must be specified in the order in which the result sets will be returned by the stored procedure invocation.

Parameters:

procedureName - name of the stored procedure in the database

resultSetMappings - the names of the result set mappings to be used in mapping result sets returned by the stored procedure

Returns:

the new stored procedure query instance

Throws:

IllegalArgumentException - if a stored procedure or result set mapping of the given name does not exist (or the query execution will fail)

joinTransaction

```
void joinTransaction()
```

Indicate to the entity manager that a JTA transaction is active and join the persistence context to it.

This method should be called on a JTA application managed entity manager that was created outside the scope of the active transaction or on an entity manager of type SynchronizationType.UNSYNCHRONIZED to associate it with the current JTA transaction.

Throws:

TransactionRequiredException - if there is no transaction

isJoinedToTransaction

```
boolean isJoinedToTransaction()
```

Determine whether the entity manager is joined to the current transaction. Returns false if the entity manager is not joined to the current transaction or if no transaction is active

Returns:

boolean

Since:

Java Persistence 2.1

unwrap

```
<T> T unwrap(Class<T> cls)
```

Return an object of the specified type to allow access to the provider-specific API. If the provider's EntityManager implementation does not support the specified class, the PersistenceException is thrown.

Parameters:

cls - the class of the object to be returned. This is normally either the underlying EntityManager implementation class or an interface that it implements.

Returns:

an instance of the specified class

Throws:

PersistenceException - if the provider does not support the call

Since:

Java Persistence 2.0

getDelegate

```
Object getDelegate()
```

Return the underlying provider object for the `EntityManager`, if available. The result of this method is implementation specific.

The `unwrap` method is to be preferred for new applications.

Returns:

`underlying provider object for EntityManager`

close

```
void close()
```

Close an application-managed entity manager. After the close method has been invoked, all methods on the `EntityManager` instance and any `Query`, `TypedQuery`, and `StoredProcedureQuery` objects obtained from it will throw the `IllegalStateException` except for `getProperties`, `getTransaction`, and `isOpen` (which will return false). If this method is called when the entity manager is joined to an active transaction, the persistence context remains managed until the transaction completes.

Throws:

`IllegalStateException - if the entity manager is container-managed`

isOpen

```
boolean isOpen()
```

Determine whether the entity manager is open.

Returns:

`true until the entity manager has been closed`

getTransaction

```
EntityTransaction getTransaction()
```

Return the resource-level `EntityTransaction` object. The `EntityTransaction` instance may be used serially to begin and commit multiple transactions.

Returns:

`EntityTransaction instance`

Throws:

`IllegalStateException - if invoked on a JTA entity manager`

getEntityManagerFactory

```
EntityManagerFactory getEntityManagerFactory()
```

Return the entity manager factory for the entity manager.

Returns:

EntityManagerFactory instance

Throws:

IllegalStateException - if the entity manager has been closed

Since:

Java Persistence 2.0

getCriteriaBuilder

```
CriteriaBuilder getCriteriaBuilder()
```

Return an instance of CriteriaBuilder for the creation of CriteriaQuery objects.

Returns:

CriteriaBuilder instance

Throws:

IllegalStateException - if the entity manager has been closed

Since:

Java Persistence 2.0

getMetamodel

```
Metamodel getMetamodel()
```

Return an instance of Metamodel interface for access to the metamodel of the persistence unit.

Returns:

Metamodel instance

Throws:

IllegalStateException - if the entity manager has been closed

Since:

Java Persistence 2.0

createEntityGraph

```
<T> EntityGraph<T> createEntityGraph(Class<T> rootType)
```

Return a mutable EntityGraph that can be used to dynamically create an EntityGraph.

Parameters:

rootType - class of entity graph

Returns:

entity graph

Since:

Java Persistence 2.1

createEntityGraph

```
EntityGraph<?> createEntityGraph(String graphName)
```

Return a mutable copy of the named EntityGraph. If there is no entity graph with the specified name, null is returned.

Parameters:

graphName - name of an entity graph

Returns:

entity graph

Since:

Java Persistence 2.1

getEntityGraph

```
EntityGraph<?> getEntityGraph(String graphName)
```

Return a named EntityGraph. The returned EntityGraph should be considered immutable.

Parameters:

graphName - name of an existing entity graph

Returns:

named entity graph

Throws:

IllegalArgumentException - if there is no EntityGraph of the given name

Since:

Java Persistence 2.1

getEntityGraphs

```
<T> List<EntityGraph<? super T>> getEntityGraphs(Class<T> entityClass)
```

Return all named EntityGraphs that have been defined for the provided class type.

Parameters:

entityClass - entity class

Returns:

list of all entity graphs defined for the entity

Throws:

IllegalArgumentException - if the class is not an entity

Since:

Java Persistence 2.1