Manipulating JPA Entities With Entity Manager API

After completing this lesson, you should be able to do the following:

➤ Declare an `EntityManager` reference with the `@PersistenceContext` annotation

➤ Look up an `EntityManager` reference by using dependency injection

➤ Use the `EntityManager` API to:

   – Find an entity by its primary key

   – Insert a new entity

   – Modify an existing entity

   – Delete an entity

➤ Execute dynamic queries with the `Query` API
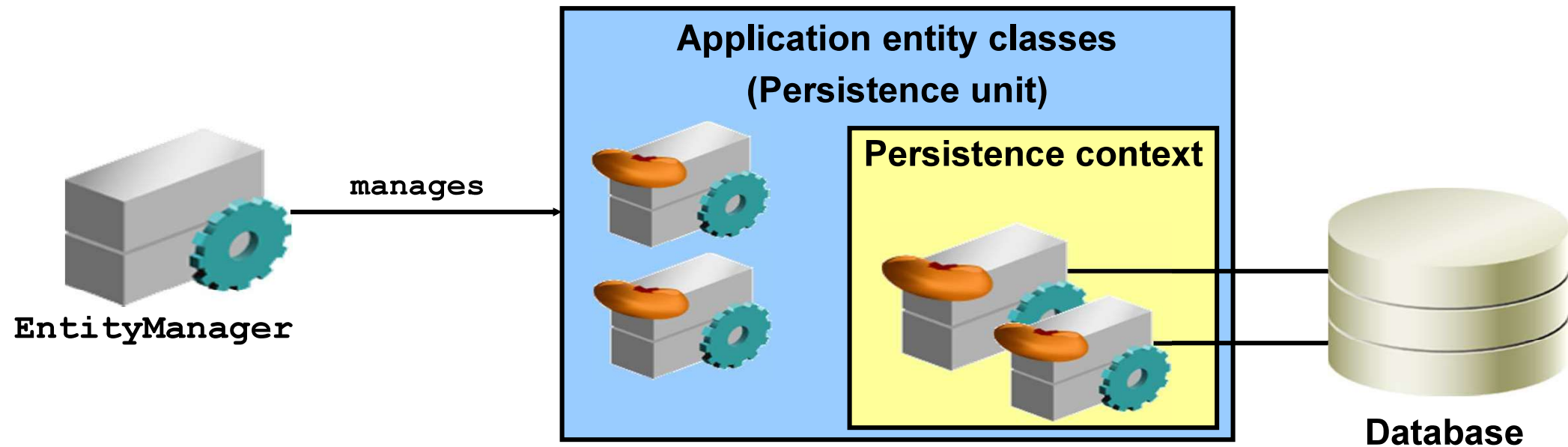
➤ Write simple JPQL queries

`EntityManager`:

➢ Is an interface defined in JPA

➢ Is a standard API for performing CRUD operations for entities

➢ Acts as a bridge between the object-oriented and the relational models
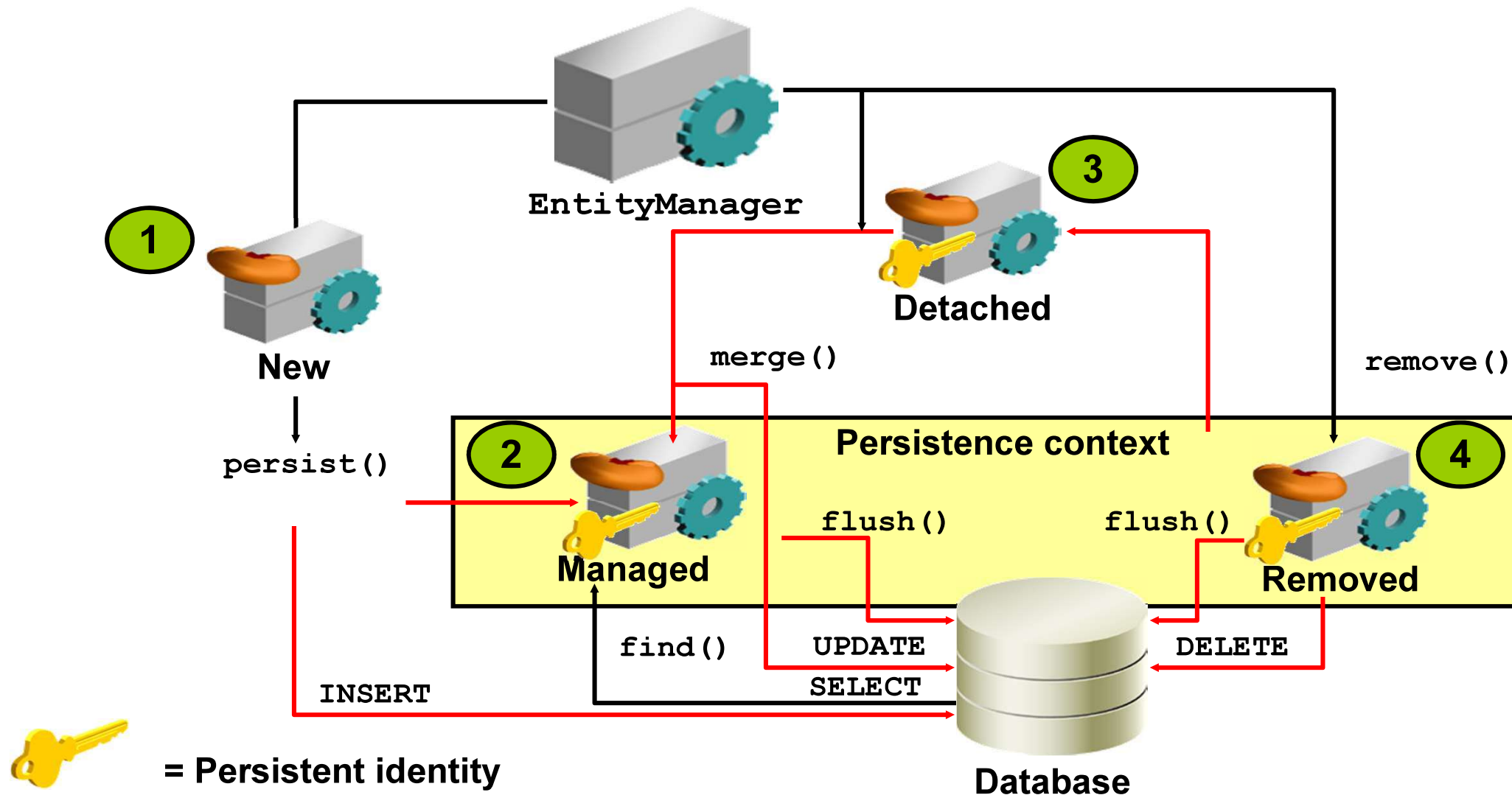

EntityManager   **manages** →   **Entities**
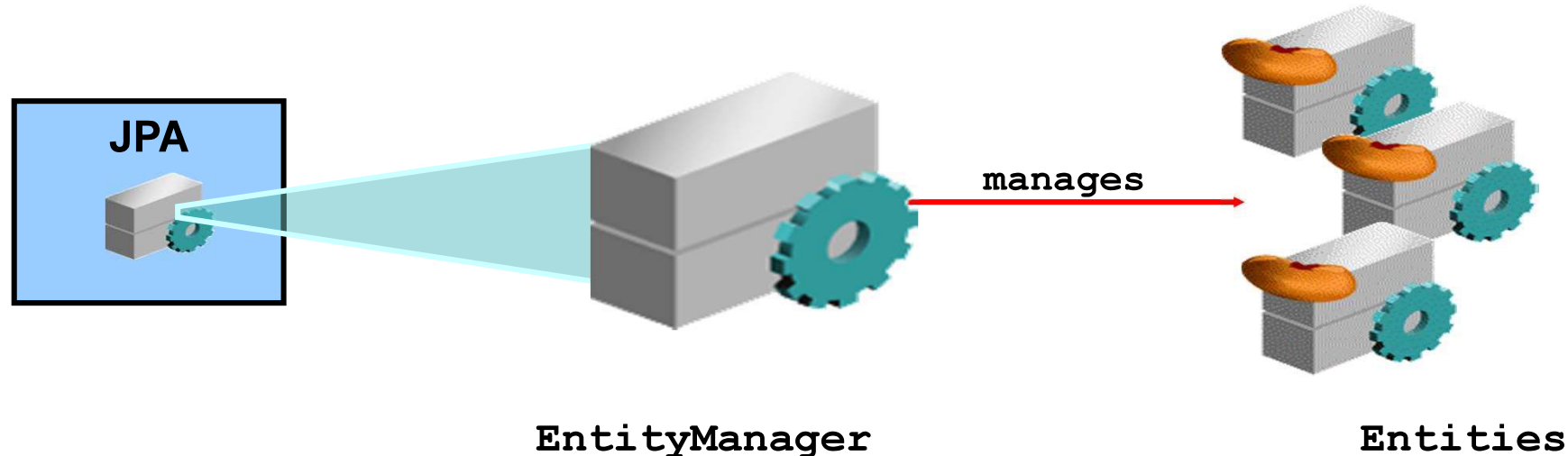
What Is `EntityManager`?

`EntityManager` is:

➢ Associated with a persistence context

➢ An object that manages a set of entities defined by a persistence unit

EntityManager manages → **Application entity classes (Persistence unit)** / **Persistence context** / **Database**

Managing an Entity Life Cycle

EntityManager

1 New

2 Managed

3 Detached

4 Removed

Persistence context

persist()

merge()

remove()

flush()

flush()

find()

UPDATE

SELECT

DELETE

INSERT

= Persistent identity

Database

Accessing an `EntityManager` Instance

➢ Container-managed `EntityManager` instances:

– Are implemented inside a Java EE container

– Use the `@PersistenceContext` annotation

– Are obtained in an application through dependency injection or JNDI lookup

➢ Application-managed `EntityManager` instances:

– Are implemented outside a Java EE container

– Are obtainable by using the `EntityManagerFactory` interface

A session bean using container injection:

```
@Stateless

public class CustomerBean {
    @PersistenceContext(unitName="FOD")

    private EntityManager em;

    ...

    public void createCustomer() {
        final Customer cust = new Customer();

        cust.setName("Valli Pataballa");

        ...

        em.persist(cust);

    }

    ...

}
```
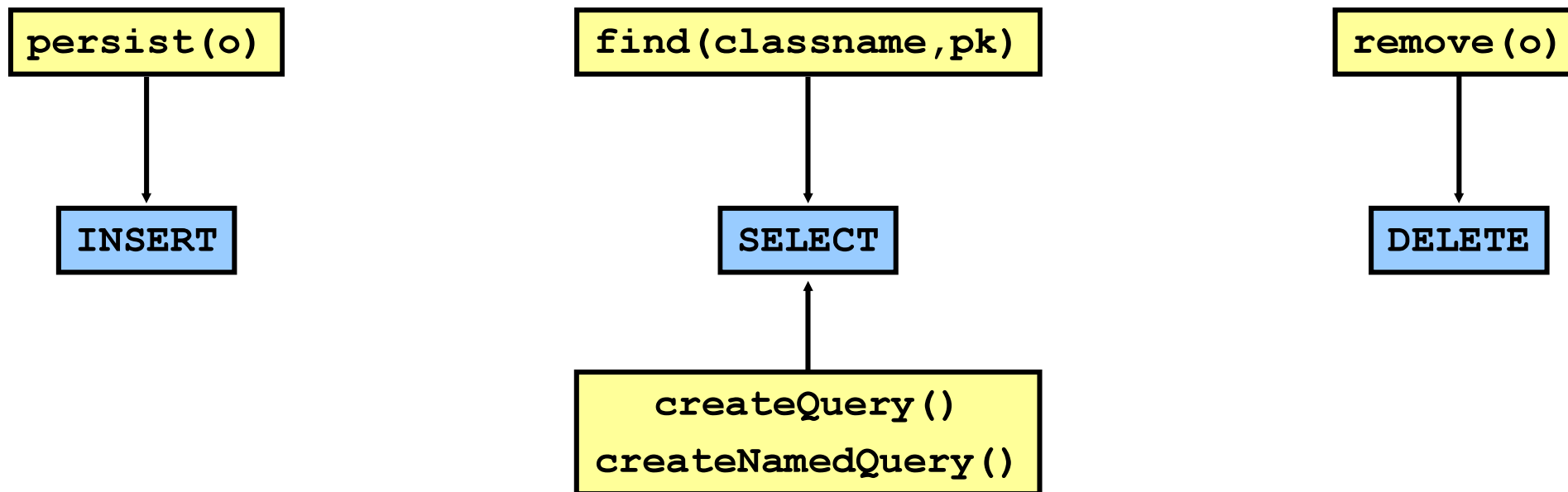
Creating an Application-Managed `EntityManager`

A Java SE application using an `EntityManagerFactory` API:

```java
public class CustomerApp {
    public static void main(String args[]) {
        final EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("FOD");
        final EntityManager em = emf.createEntityManager();
        final Customer cust = new Customer()
        cust.setName("Ron Howard");

        ...

        em.persist(cust);


    }
}
```

MentorLabs℠

The `EntityManager` API provides the following methods that map to CRUD database operations:

```
persist(o)            find(classname,pk)            remove(o)
```

```
INSERT                SELECT                        DELETE
```

```
createQuery()
createNamedQuery()
```

Method signature of the most commonly used methods of the `EntityManager` interface:

```
public void persist(Object entity);
public <T> T merge(T entity);
public void remove(Object entity);
public <T> T find(class<T> entityClass, Object
                                    primaryKey);
public void flush();
public void refresh(Object entity);
public void clear();
public Query createQuery (String jpqlString);
public Query createNamedQuery (String name);
```

MENTORLABS℠

The application-managed `EntityManager` is implemented outside a Java EE container.

1. True

2. False

To insert new data, perform the following steps:

1. Create a new entity object.

2. Call the `EntityManager.persist()` method.

```java
@PersistenceContext(unitName="Model")
private EntityManager em; // inject the EntityManager
...                       // object
public void persistUser() {
  Users user = new Users();
  user.setFirstName("Steve");
  user.setLastName("King");

  em.persist(user);
  // On return the user object contains persisted state
  // including fields populated with generated id values
}
```

To delete data, perform the following steps:

1. Find, set, or refresh the state of the entity to be deleted.

2. Call the `EntityManager.remove()` method.

```java
@PersistenceContext(unitName="Model")

private EntityManager em;

...

// Remove a Product by primary key Id value
 public void removeProducts(Products products) {
   products = em.find(Products.class,
                              products.getProdId());
   em.remove(products);
 }
...
```

Updating and Synchronizing the Entity with the Database

1. Retrieve the rows in a database table into the entity by using the `EntityManager` API.

2. Use the setter methods of the entity to update the data attributes.

3. Use the `flush()` method to synchronize the state of the entities in the `EntityManager`'s persistence context with the database.

4. Use the `merge()` method to reattach an entity to the persistence context to synchronize it with the database.

MENTORLABS℠

To update data, perform the following steps:

1. Find, set, or refresh the state of the entity to be updated.

2. Call the `EntityManager.merge()` method.

```java
@PersistenceContext(unitName="Model")
private EntityManager em;
...
// Update a Product by primary key Id value
 public void updateProducts(Products products) {
   products = em.find(Products.class,
                              products.getProdId());
   products.setListPrice("1000");
...
   em.merge(products);
 }
...
```

Finding an Entity by Primary Key

To locate an entity by primary key, perform the following steps:

1. Create and set the primary key object and value.

2. Call the `EntityManager find()` method with the following parameters:

   – Entity class

   – Primary-key object

```java
import org.srdemo.persistence.Users;
@PersistenceContext(unitName="Model")
private EntityManager em;
...
public Users findUserByPrimaryKey(Long id) {
  Users user = null;
  user = em.find(Users.class, id);
  return user;
}
```

The `EntityManager.flush()` method:

1. Retrieves the rows in a database table into the entity

2. Reattaches an entity to the persistence context

3. Synchronizes the state of the entity in the `EntityManager`'s persistence context with the database
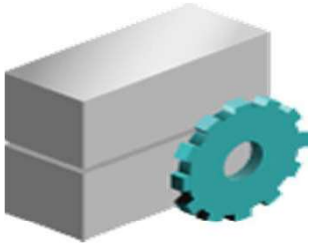
What Is JPA `Query` API?

The JPA `Query` API:

➢ Includes:

– `EntityManager` methods to create queries

– `Query` interface methods for executing queries

– Java Persistence Query Language (JPQL)

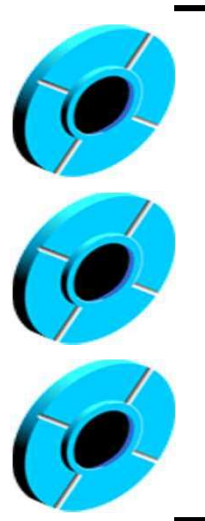➢ Supports:

– Named queries

– Dynamic queries

The `EntityManager` interface provides the `Query` API methods to execute JPQL statements:

EntityManager

createQuery(String jpql)

**createNamedQuery(
    String name)**

Query instance methods:

setParameter(String, Object)

Object getSingleResult()

List getResultList()

Query setMaxResults(int)

Query setFirstResult(int)

int executeUpdate()

To create a named query, perform the following steps:

1. Define the query with the `NamedQuery` annotation.

```java
@NamedQuery(name="findUsersByCity",
    query="SELECT object(o) FROM Users o " +
        "where o.city = :city");
```

2. Create a `Query` object for the named query with the `createNamedQuery()` method, setting parameters and returning results.

```java
public List<Users> findUsersinCity(String cityName) {
  Query query = em.createNamedQuery("findUsersByCity");
  query.setParameter("city", cityName);
  return query.getResultList();
}
```

Example: Find service requests by primary key and a specified status.

```
public List findServiceRequests(Long id, String status){
  if (id != null && status != null ) {
    Query query = em.createQuery(
     "select object(sr) from ServiceRequests sr " +
     "where sr.svrId = :srvId and sr.status = :status");
    query.setParameter("srvId", id);
    query.setParameter("status", status);
    return query.getResultList();
  }
  return null;
}
```

MENTORLABS℠

In this lesson, you should have learned how to:

➢ Declare an `EntityManager` reference with the `@PersistenceContext` annotation

➢ Look up an `EntityManager` reference by using dependency injection

➢ Use the `EntityManager` API to:

– Find an entity by its primary key

– Insert a new entity

– Modify an existing entity

– Delete an entity

➢ Execute dynamic queries with the `Query` API

➢ Write simple JPQL queries

MENTORLABS