LUXOFT TRAINING

JVA-000
Java Persistence with Hibernate

Module 3
Entity Manager

Objectives

- Observe `EntityManager` API
- Learn how to use `EntityManager` to manage the entity instance lifecycle

EntityManager API

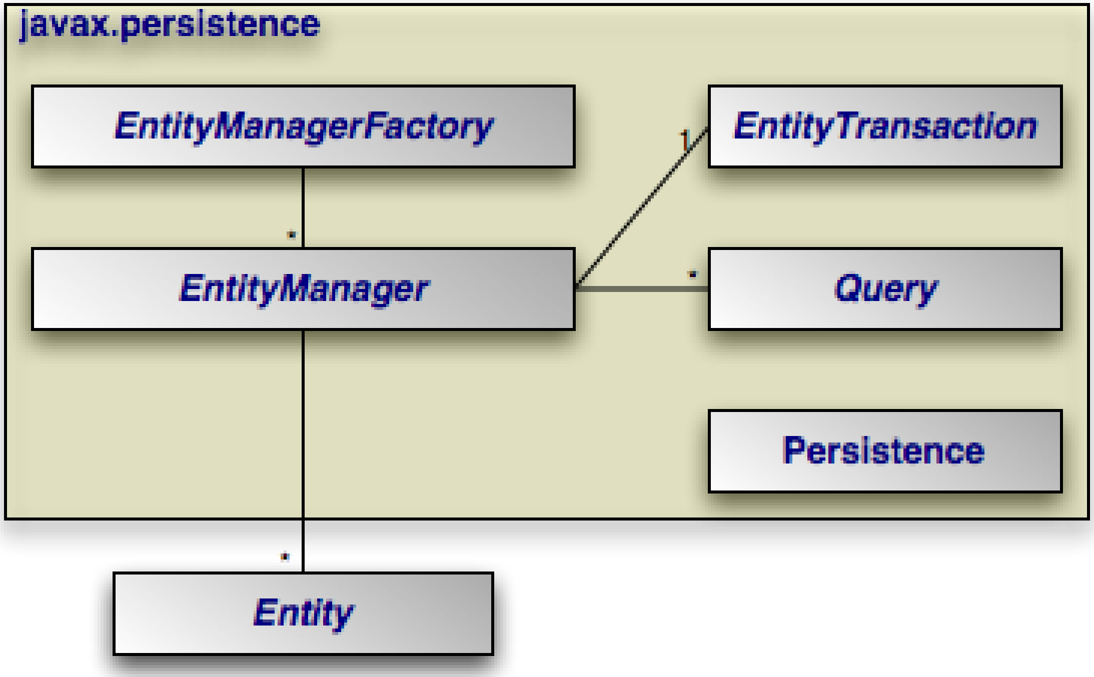
`EntityManager` is an interface that **defines methods to interact with the persistence context**.

Persistence context is a **set of entity instances** in which for any entity identity there is a **unique entity instance**.

Within the persistence context, the entity instances and their lifecycle are managed.

The set of entities that can be managed by a given `EntityManager` instance is defined by a **persistence unit**.

EntityManager API



Relationships between the primary components of the JPA architecture

EntityManager API

`EntityManager` defines the following methods to manage entities:

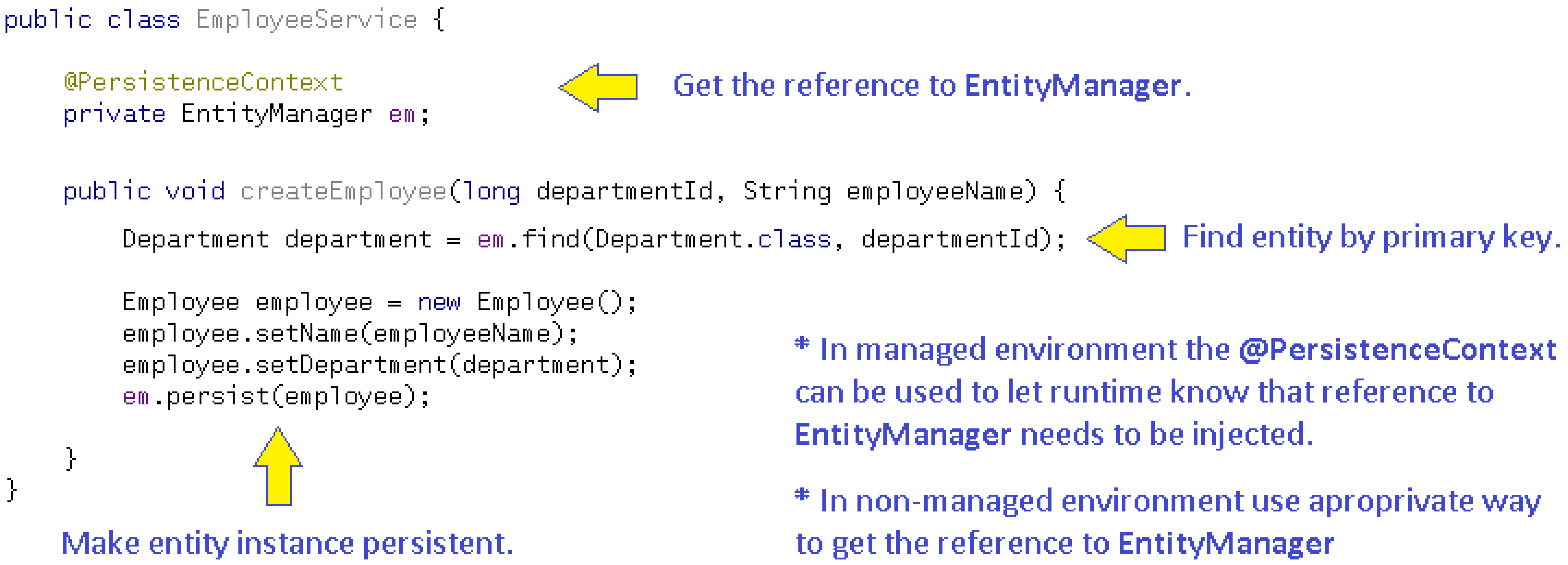
- **`persist()`** – to make an entity instance managed and persistent.
- **`merge()`** – to merge the state of an entity into the current persistence context.
- **`remove()`** – to remove an entity instance.
- **`find()`** – to find an entity by primary key.
- **`lock()`** – to lock an entity instance that is contained in the persistence context with the specified lock mode type.

EntityManager API

`EntityManager` defines the following methods to manage entities *(continue)*:

- **`refresh()`** – to refresh the state of the instance from the database, overwriting changes made to the entity, if any.
- **`detach()`** – to remove an 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.*

EntityManager API



```
public class EmployeeService {

    @PersistenceContext
    private EntityManager em;

    public void createEmployee(long departmentId, String employeeName) {
        Department department = em.find(Department.class, departmentId);

        Employee employee = new Employee();
        employee.setName(employeeName);
        employee.setDepartment(department);
        em.persist(employee);

    }
}
```

Get the reference to **EntityManager**.

Find entity by primary key.

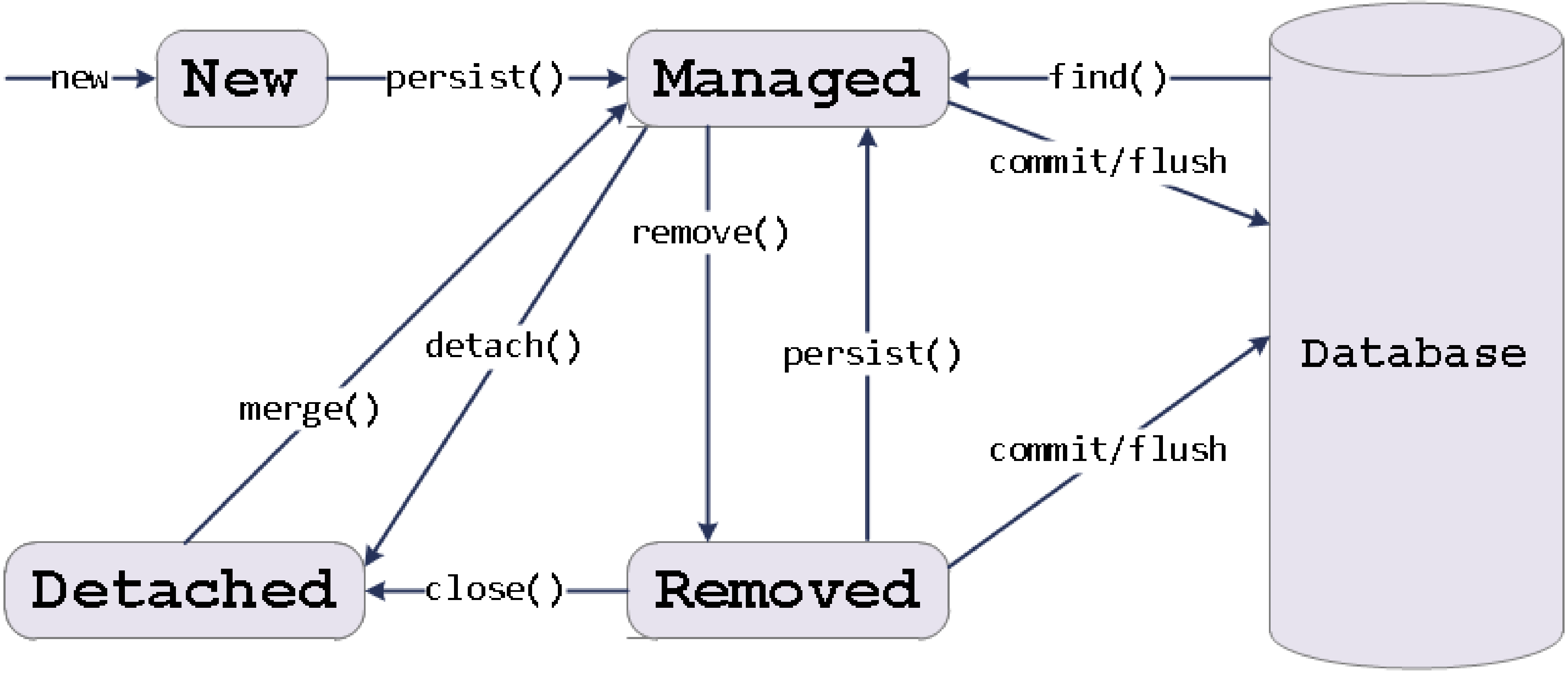
Make entity instance persistent.

* In managed environment the **@PersistenceContext** can be used to let runtime know that reference to **EntityManager** needs to be injected.

* In non-managed environment use aproprivate way to get the reference to **EntityManager**

Example of use of `EntityManager` to handle manage entities

Entity Life Cycle



Entity states and transitions between them

Entity Life Cycle: `persist()` method

Calling `persist()` method results in (*state* → *result*):

- **New** → entity becomes managed
- **Managed** → operation ignored for the entity and cascaded to referenced entities
- **Removed** → entity becomes managed
- **Detached** → exception thrown (right after operation is called or after transaction is flushed/committed)

Entity Life Cycle: `remove()` method

Calling `remove()` method results in *(state → result)*:

- **New →** ignored
- **Managed →** entity becomes Removed, operation cascades to referenced entities
- **Removed →** ignored
- **Detached →** exception thrown

Date are removed from DB after transaction commit or flush operation gets called.

Entity Life Cycle: `refresh()` method

Calling `refresh()` method results in (*state* → *result*):

- **Managed** → entity reloaded from DB, operation cascaded to referenced entities
- **New/Detached/Removed** → exception thrown

Entity Life Cycle: `merge()` method

Calling `merge()` method results in *(state → result)*:

- **New** → new instance created, stated of merged entity copied into it, and entity becomes managed.
- **Managed** → ignored, operation cascaded to referenced entities
- **Detached** → existing entity loaded, state of merged entity copied into it
- **Removed** → exception thrown

Entity Life Cycle: Sync. to DB

The state of persistent entities is **synchronized to the database at transaction commit or flush operation**.

Synchronization to database **doesn't involve refresh** of any managed entities unless the `refresh()` is explicitly invoked or cascaded to them.

Bidirectional relationships between entities will be **persisted based on references held by the owning side** of relationship *(it's developer's responsibility to keep in-memory references consistent)*.

Context of type **`UNSYNCHRONIZED`** will be **persisted after joining** it **to the current transaction**

Persistence Context Types

Persistence context can be of types:

- Application-managed
- Container-managed

The lifetime of a container-managed persistence context can either:

- Be scoped to a transaction (**transaction-scoped context**).
- Have a lifetime scope that **extends** beyond of single transaction (**extended persistence context**)

`EntityManager` with extended persistence context keeps entities managed after transaction commit

Persistence Context Types

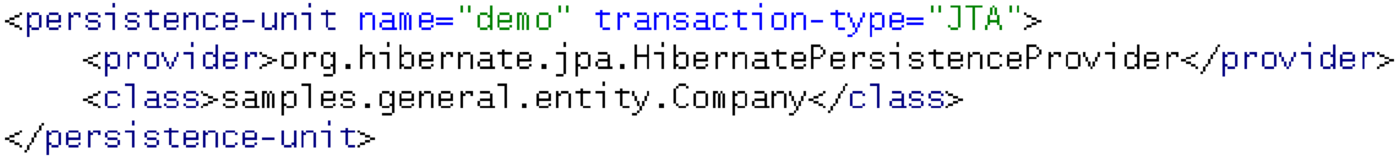
In case of `EntityManager` with extended persistence context the `persist()`, `remove()`, `merge()` and `refresh()` methods can be called even there is no active transaction

Effects of these operations will be committed to the database when the extended persistence context is enlisted in a transaction and the transaction commits

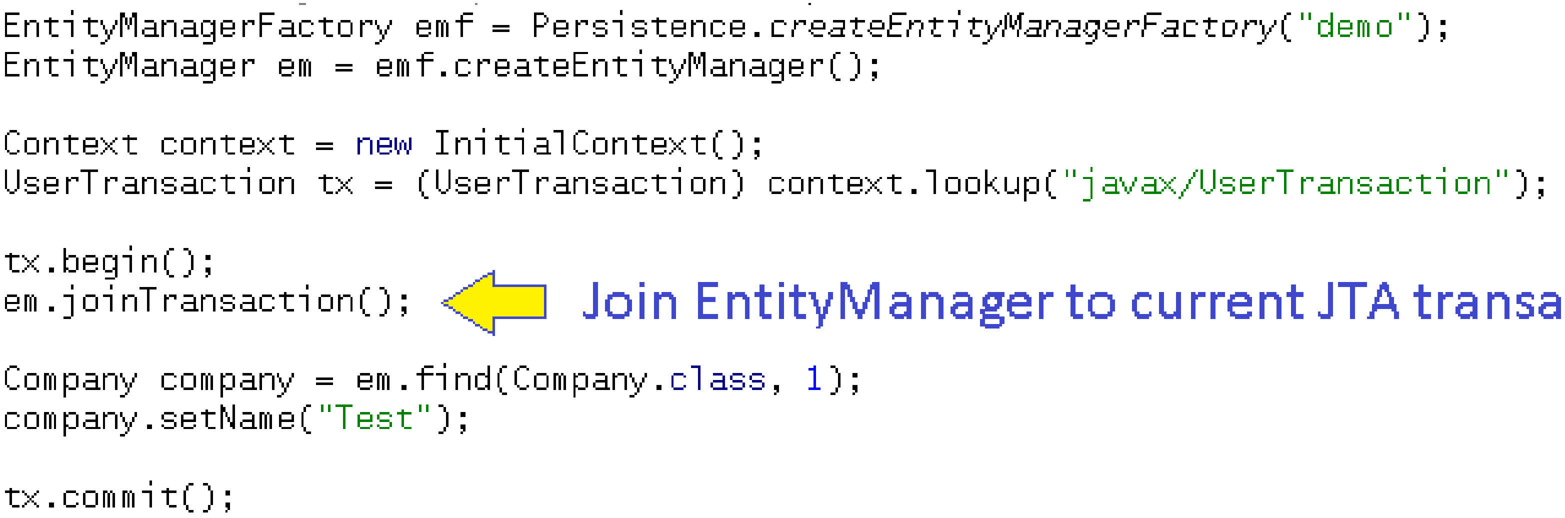
Use method `EntityManager.joinTransaction()` to join persistence context to current JTA transaction

Persistence Context Types

Transaction type should be JTA



```
<persistence-unit name="demo" transaction-type="JTA">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <class>samples.general.entity.Company</class>
</persistence-unit>
```



```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("demo");
EntityManager em = emf.createEntityManager();

Context context = new InitialContext();
UserTransaction tx = (UserTransaction) context.lookup("javax/UserTransaction");

tx.begin();
em.joinTransaction();

Company company = em.find(Company.class, 1);
company.setName("Test");

tx.commit();
```

Join EntityManager to current JTA transaction

`EntityManager.joinTransaction()` allows an application managed `EntityManager` to join the active JTA transaction context

Obtaining an `EntityManager`

The `EntityManager` for a persistence context is obtained from an entity manager factory.

When container-managed `EntityManager` is used (Java EE environment):

- Application doesn't interact with the entity manager factory
- `EntityManager` is obtained through dependency injection or from JNDI *(container manages interaction with the entity manager factory transparently to the application)*

When application-managed `EntityManagers` is used, the application use the entity manager factory to create the entity manager.

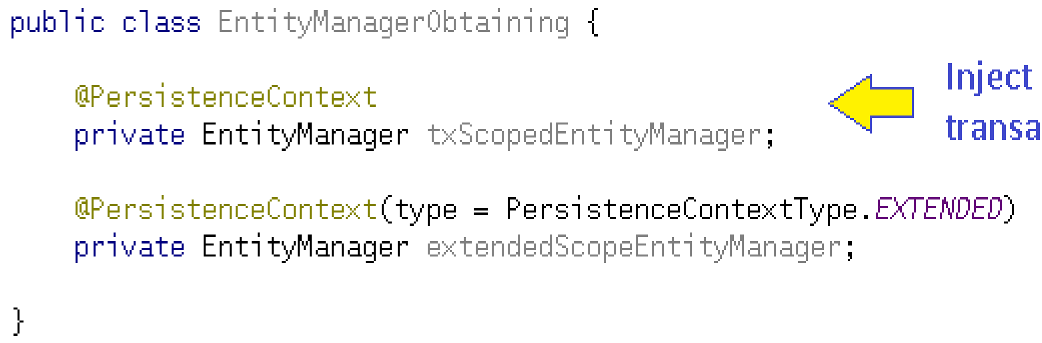
Obtaining container-managed `EntityManager`

Container-managed `EntityManager` is obtained either:

- Through dependency injection
- Through lookup of the entity manager in the JNDI namespace

`@PersistenceContext` annotation is used to inject reference to `EntityManager`

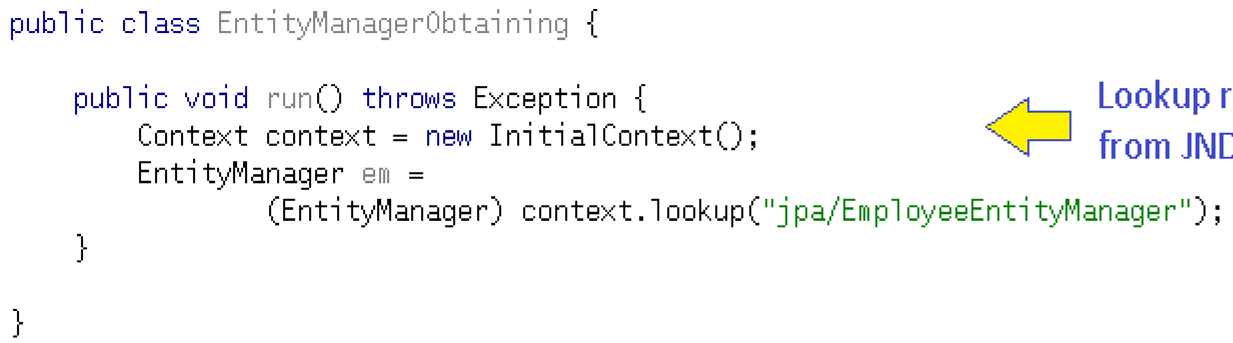
Obtaining container-managed EntityManager



```
public class EntityManagerObtaining {

    @PersistenceContext
    private EntityManager txScopedEntityManager;

    @PersistenceContext(type = PersistenceContextType.EXTENDED)
    private EntityManager extendedScopeEntityManager;

}
```

Inject reference to
transaction scoped EntityManager

Inject reference to
EntityManager with extend scope

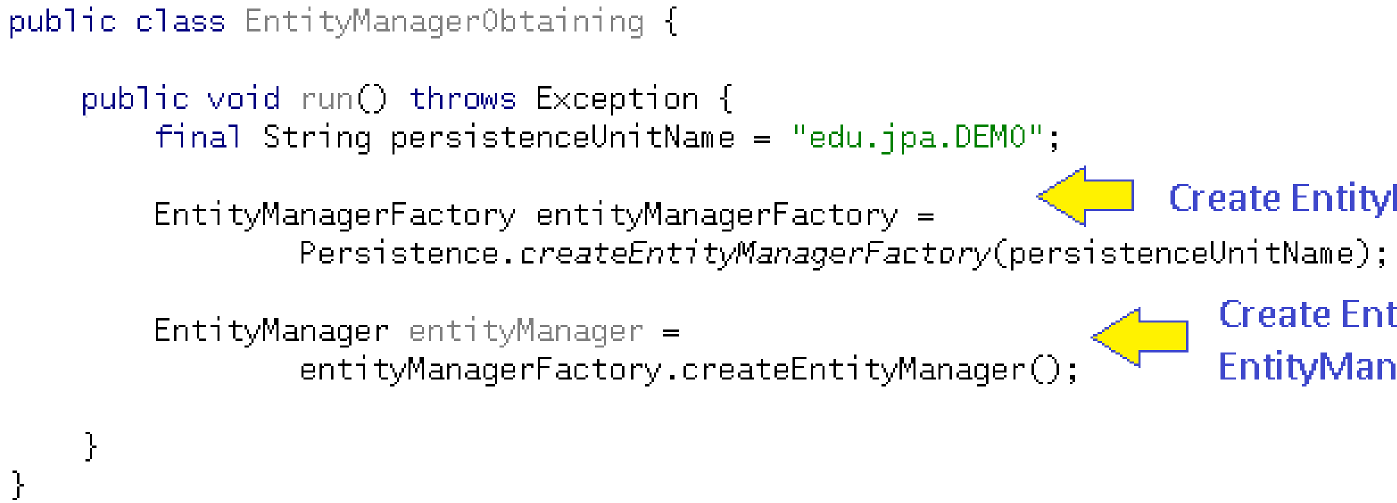


```
public class EntityManagerObtaining {

    public void run() throws Exception {
        Context context = new InitialContext();
        EntityManager em =
                (EntityManager) context.lookup("jpa/EmployeeEntityManager");
    }

}
```

Lookup reference to EntityManager
from JNDI service

Example of how to obtain reference to container-managed EntityManager in Java EE environment

Obtaining container-managed EntityManager



```
public class EntityManagerObtaining {

    public void run() throws Exception {
        final String persistenceUnitName = "edu.jpa.DEMO";

        EntityManagerFactory entityManagerFactory =
                Persistence.createEntityManagerFactory(persistenceUnitName);

        EntityManager entityManager =
                entityManagerFactory.createEntityManager();

    }
}
```

Create EntityManager factory

Create EntityManager using EntityManager factory

Example of how to obtain reference to application-managed EntityManager in Java SE environment

`EntityManagerFactory` Interface

The `EntityManagerFactory` interface is used to obtain an application-managed entity manager.

The entity manager factory should be closed before application stops working

Once entity manager factory is closed all `EntityManager` instances are considered as closed

Provides access to the second-level cache maintained by persistence provider (`javax.persistence.Cache`)

EntityManagerFactory Interface

EntityManagerFactory methods:

- createEntityManager() – creates a new application-managed EntityManager
- getMetamodel() – returns instance of Metamodel interface for access to the meta-model of the persistence unit
- isOpen() – indicates whether the factory is open
- close() – closes the factory, releasing any resources that it holds
- getCache() – returns the cache that is associated with the entity manager factory (the “second level cache”)
- getProperties() – returns the properties that are in effect for the entity manager factory *(read only, changing has no effect)*

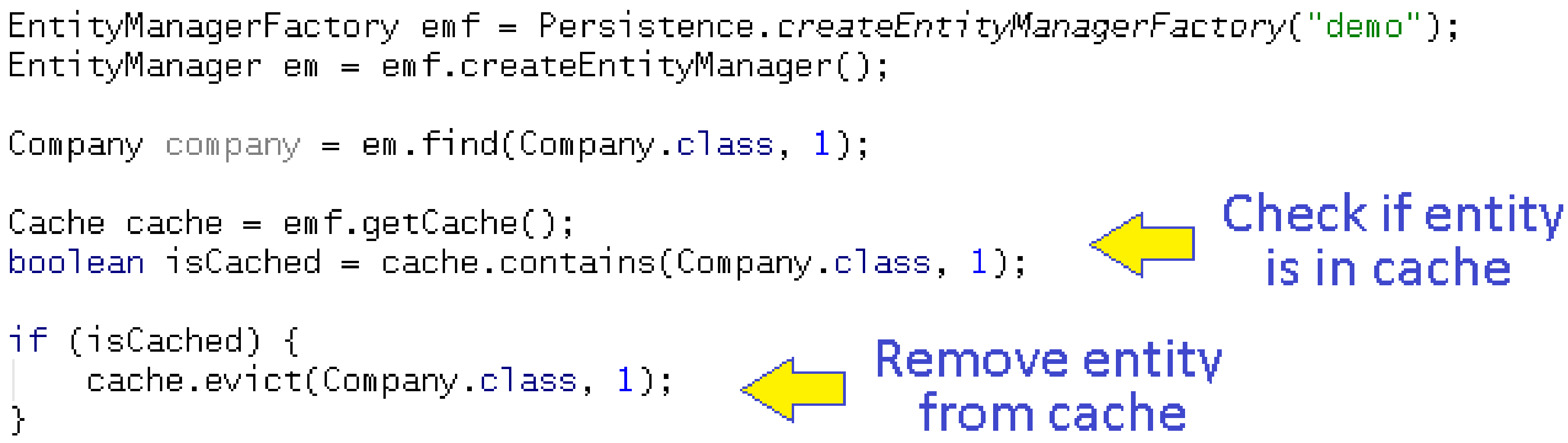
Cache Interface

`javax.persistence.Cache` interface **provides** basic **functionality over** the persistence provider's **second level cache** *(if used)*.

Methods:

- `contains(Class, Object)` – checks whether the cache contains data for the given entity
- `evict(Class, Object)` – removes the data for the given entity from the cache
- `evict(Class)` – removes the data for entities of the specified class *(and its subclasses)* from the cache
- `evictAll()` – clears the cache

Cache Interface



```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("demo");
EntityManager em = emf.createEntityManager();

Company company = em.find(Company.class, 1);

Cache cache = emf.getCache();
boolean isCached = cache.contains(Company.class, 1);

if (isCached) {
    cache.evict(Company.class, 1);
}
```

Check if entity is in cache

Remove entity from cache

Here is an example of accessing 2nd level cache via Cache interface.

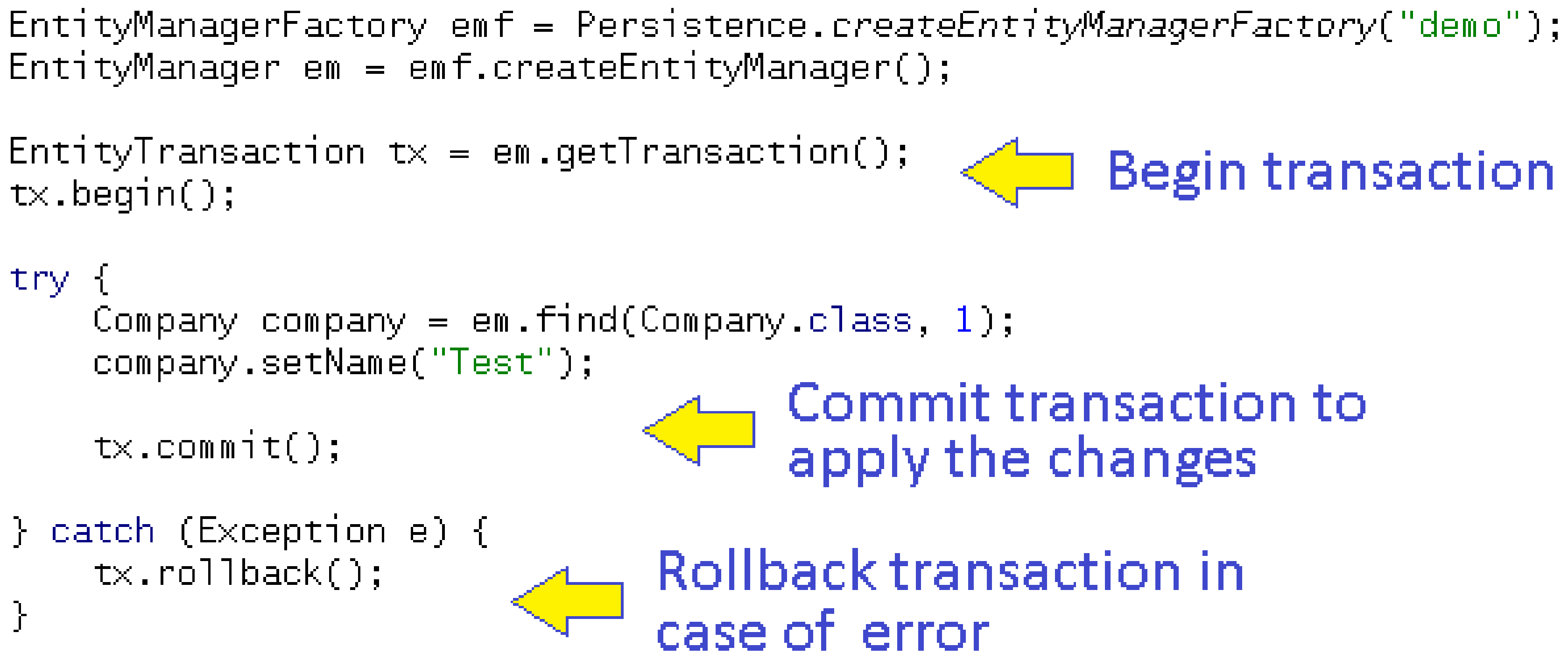
`EntityTransaction` Interface

`javax.persistence.EntityTransaction` interface provides basic functionality to **control local resource transactions**

Methods:

- `begin()` - starts a resource transaction
- `commit()` - commits the current resource transaction
- `rollback()` - roll back the current resource transaction
- `setRollbackOnly()` - marks transaction as read only
- `getRollbackOnly()` - determines whether the current transaction has been marked for rollback
- `isActive()` - indicates whether transaction is in progress

EntityTransaction Interface



```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("demo");
EntityManager em = emf.createEntityManager();

EntityTransaction tx = em.getTransaction();
tx.begin();

try {
    Company company = em.find(Company.class, 1);
    company.setName("Test");

    tx.commit();

} catch (Exception e) {
    tx.rollback();
}
```

Begin transaction

Commit transaction to apply the changes

Rollback transaction in case of error

Here is an example of local transaction usage.

Local JPA transactions are defined through the `EntityTransaction` class. It contains basic transaction API including begin, commit and rollback.

Practical work

Exercise 5:

Working with EntityManager

Thank you for your attention!

Questions?