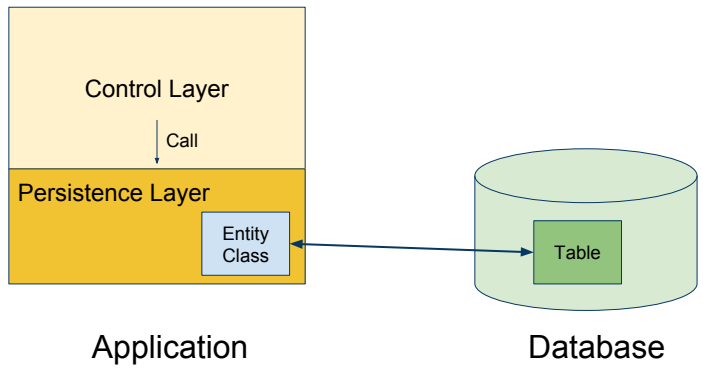
JPA Managing

JPA Slides #3

Content

- Persistence Layer
- Enterprise Java Beans
- Session Beans
- More Injection
- Persistence context
- Entity Class Life Cycle
- Cacheing, fetching and orphan removal
- Persistence and Testing

Developing a Persistence Layer



Entity classes not managed by themselves

- Handled by a persistence layer
- Persistence layer must handle transaction processing!
  - Transactions fundamental for database integrity

Basic flow

1. Begin the transaction
2. Execute a set of data manipulations and/or queries
3. If no errors occur then commit the transaction and end it
4. If errors occur then rollback the transaction and end it

If program manipulate database (i.e. any write)

- Most use transactions
  - Explicit transaction demarcation possible with statements like … (tx is object representing transaction)
    - tx.begin(), tx.commit() and tx.rollback()
- Simpler to use Enterprise JavaBeans, EJB, next slide …

Enterprise JavaBeans EJB

Enterprise JavaBeans EJBs are transaction aware objects

- Classes with annotations
- Will automagically handle transactions, we don't need to code …
  - … but if special needs also possible to code (we use for some testing)
- Must run in EJB container (GlassFish)
  - Will handle life cycle of EJBs
- Services offered by EJBs
  - Automatic transaction demarcation
    - If transaction fail, automatic rollback and exception
  - Dependency injection (inferior to CDI)
  - Concurrency handling
    - An EJB (session) bean does not have to be coded as reentrant
  - Timerservice/Scheduling
    - Used for long lived business processes (may survive server crash)
      - Send an email each year: "Time for Christmas … buy, buy, buy"..
  - Web services endpoint (EJB may act as REST resource class)
  - Security
  - Aspect oriented programming, interceptors

EJB types

- Session Beans (SB)
  - Used for business logic, processes, work flows
  - Executes on behalf of a unique client, client running the session (i.e

        - session scoped)
        - Stateless session bean, no mutable application logic state (no conversational state)
            - Work done in single method call.
        - … stateful session bean
            - If need state in between method calls (avoid)
        - Also singleton beans, not covered
        - Only stateless covered here, if specific need check out stateful
        - Possibly asynchronous, we don't
- Message Driven Beans (MDB)
    - Used by Java Message Service (JMS). Asynchronous message service with high quality of service
        - Not covered...

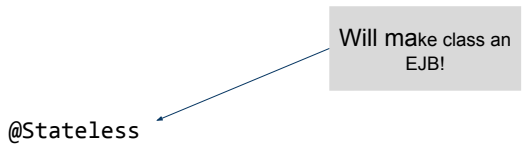
EBJs must obey
- JavaBeans specification
- Concrete class (not abstract)
- Not final
- Inheritance possibly (involving EJBs only)

EJBs should not
- use Java Reflection API to access information unavailable by way of the security rules of the Java runtime environment
- read or write non-final static fields
- use this to refer to the instance in a method parameter or result
- use the java.awt/swing packages to create a user interface
- create or modify class loaders and security managers
- redirect input, output, and error streams
- obtain security policy information for a code source
- stop the Java virtual machine…
- and more …
- … i.e. use as intended : Transaction awareness

Possible to collect EJBs in reusable packages (jar-files).
- Many front-ends to share common back-end
    - Advanced deployment issues

Much more … but we probably don't need.

Stateless EJB Sample




```
@Stateless
public class AuthorRegistry extend ... {

    @PersistenceContext
     private EntityManager em;

    public EntityManager getEntityManager() {
         return em;
     }


}
```

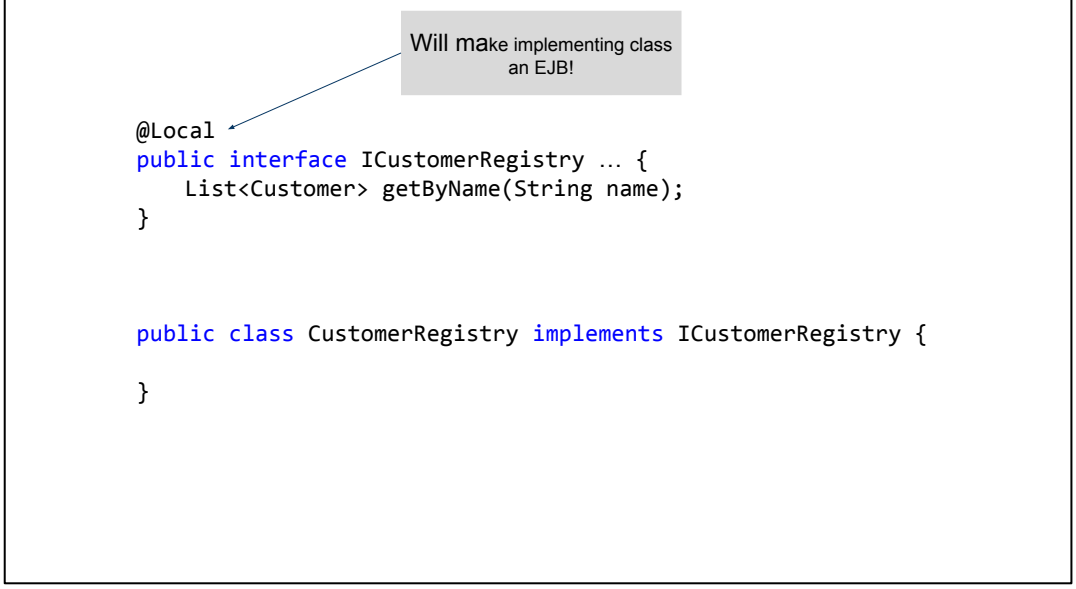
To make a POJO class an Entity class

- Add annotations (@Stateless or @Local, see next slide)

Note state is not conventional state (it's a service object)

- @PersistenceContext, upcoming

EJB using Interface

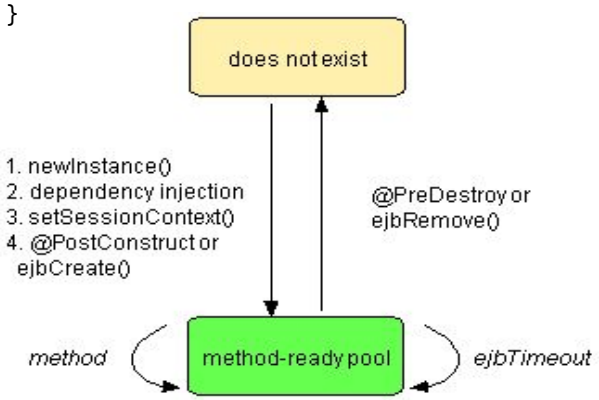


Possible to use interfaces for EJBs

- @Local, will live in same JVM
- @Remote, lives in other JVM (calls transparently over network)
  - Not used by us.

Session Bean Lifecycle

```
import javax.ejb.Stateless;
@Stateless
public class Library {
 ...
}
```



Life cycle handled by container

- Efficient, pooled by container, shared by many clients
- Callbacks at certain life cycle events, @PostConstruct, @PreDestroy

EJB Injection

```
@Named("cdibean")
@RequestScoped
public class MyCDIBean {

    @EJB
    private MyEJB myEjb;
    ...
}
```

EJB into CDI

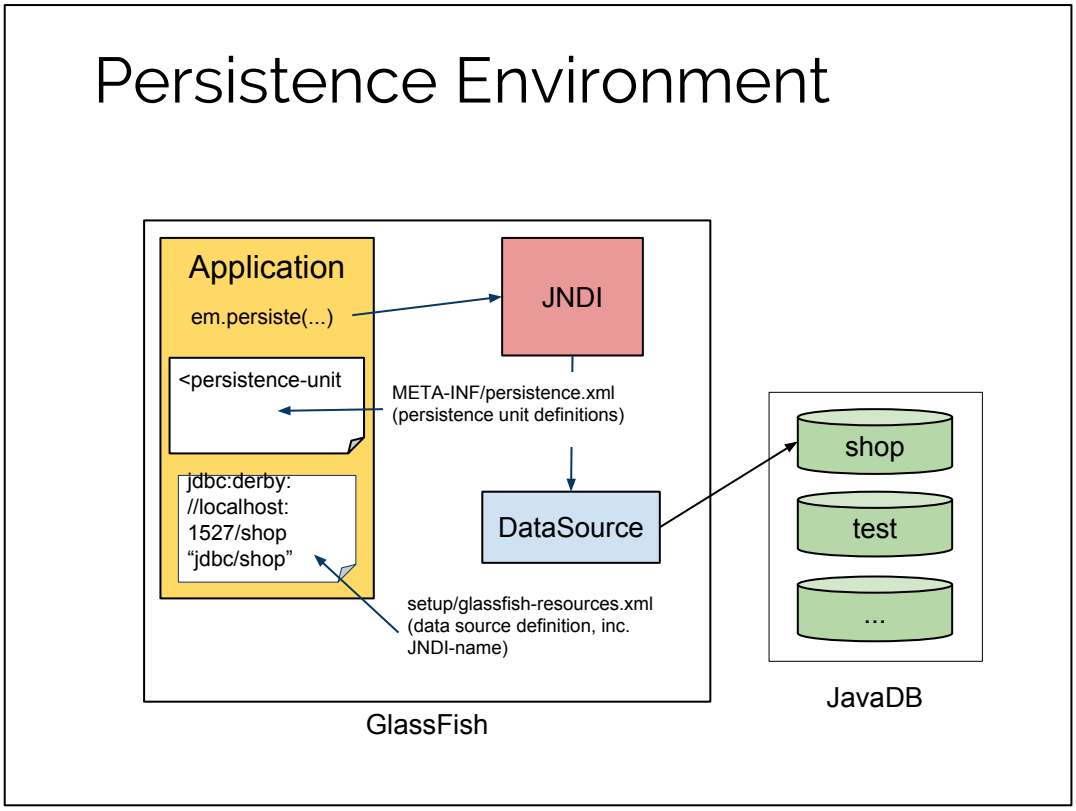
Resources	Stateless	Stateful	MDB	Interceptors
JDBC DataSource	Yes	Yes	Yes	Yes
JMS Destinations, Connection Factories	Yes	Yes	Yes	Yes
Mail Resources	Yes	Yes	Yes	Yes
UserTransaction	Yes	Yes	Yes	Yes
Environment Entries	Yes	Yes	Yes	Yes
EJBContext	Yes	Yes	Yes	No
Timer Service	Yes	No	Yes	No
Web Service reference	Yes	Yes	Yes	Yes
EnityManager, EntityManagerFactory	Yes	Yes	Yes	Yes

Resource into EJB

Dependency injection

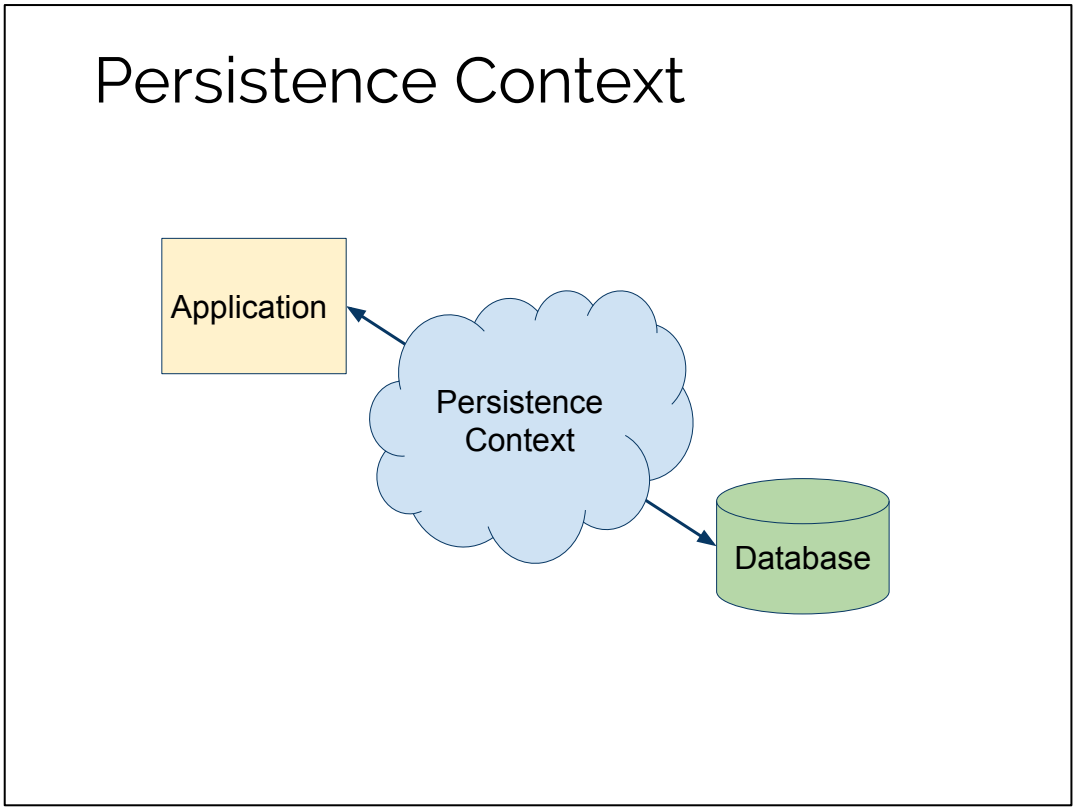
- Handled by EJB, but can coexist with CDI
- Inject EJB into CDI (or Servlet)
- Inject resources into EJB
- @Inject annotation should(!) work for EJBs too.

TIP: Use: NetBeans > Insert Code ... > Call Enterprise Bean



Tech talk

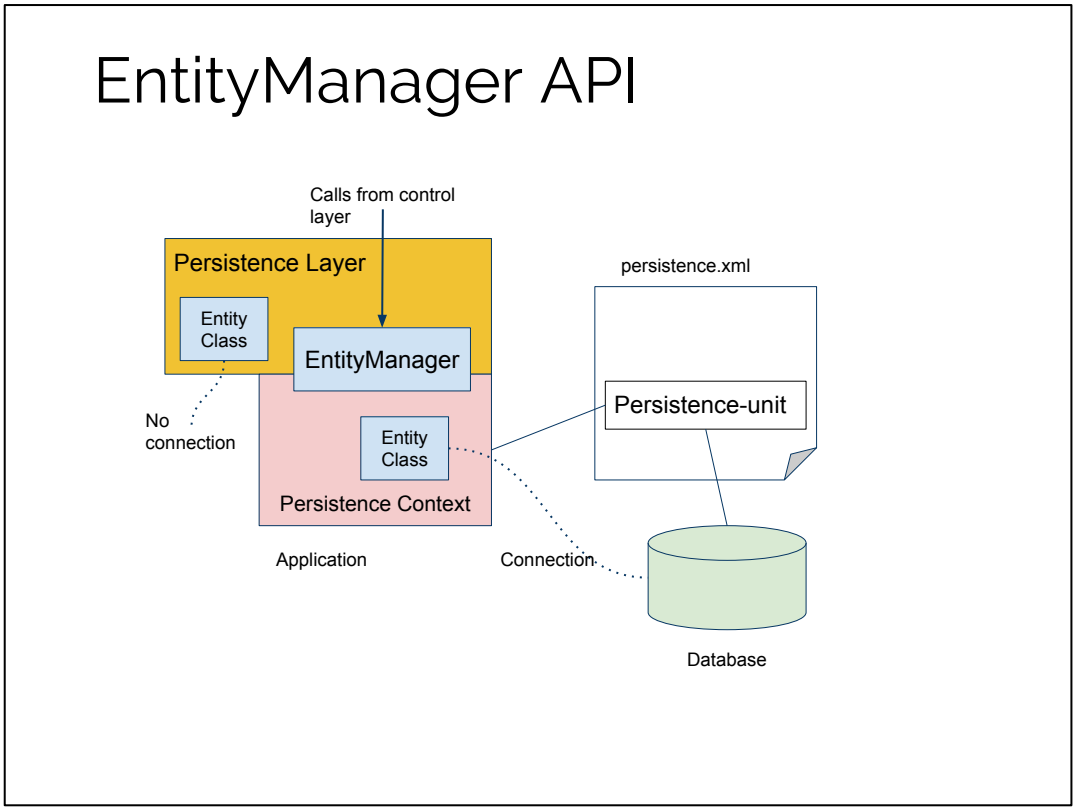
- Application runs in container (GlassFish)
- Uses JNDI, Java Naming and Directory Interface
  - Key/value-store (imagine: global Tree<String, Object>) to keep track of a …
- … DataSource object
  - Object created outside application (application external, refactored out of application)
    - Many application may use same)
    - “jdbc/shop” is the JNDI-name of the object in slide
  - Defined in glassfish-resources.xml.
    - Object holds info about database: URL (location, port), name, passwords and more
  - Creation/managing handled by NetBeans/GlassFish



Persistence Context, PC

- A set of managed entity instances (objects) in which for any persistent entity identity [primary key] there is a unique entity instance.
  - I.e. a table row (unique by primary key) will be represented as a unique in memory object in PC
    - Identity shall use ==
- Every PC is associated with a persistence unit (the list of entity classes in persistence.xml…)
- Variations
  - Transaction-Scoped: PC follows the transaction, we use!
    - Use: In stateless environment (stateless session EJB)
  - Extended: PC bound to lifecycle of some stateful managed object
    - Use: In stateful environment (stateful session EJB)
  - Both use JTA transaction (Java Transaction API, Java EE server transactions).
    - Very many customizations options for transactions in JPA, not covered

The main role of the persistence context is to make sure that a database entity object is represented by no more than one in-memory entity object within the same EntityManager (see next slide).



EntityManager API

- The entity instances and their lifecycles are managed by an EntityManager (EM) object
- Every EntityManager manages one (and only one) persistence context.
  - Therefore, a database object can be represented by different memory entity objects in different EntityManager instances. But retrieving the same database object more than once using the same EntityManager should always result in the same in-memory entity object.
  - Persistence context is a a first level cache of entities handled by the EM
- Variations:
  - Container Managed Entity Managers (container manages lifecycle of EntityManager), we use!
  - Application Managed Entity Managers (avoid)
- Possible to inject EM into EJBs

```
@PersistenceContext("PU name")  // Name optional if only one PU
private EntityManager em;
```

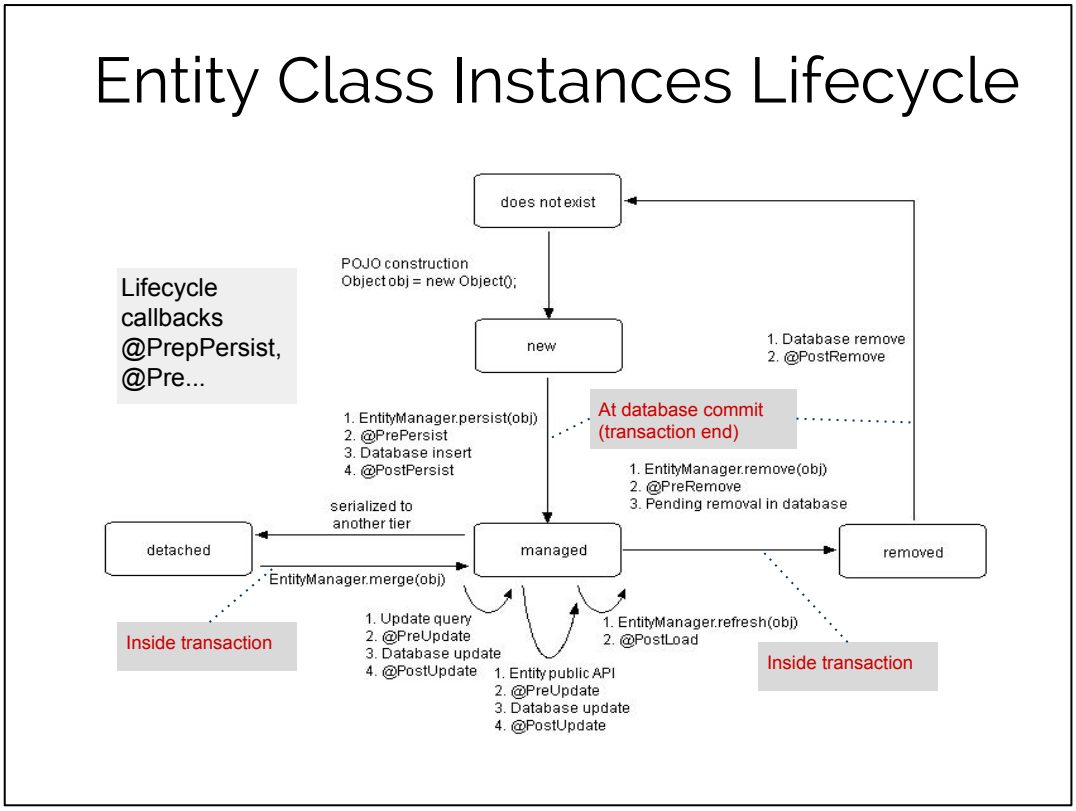
Use the EM API to interact with the persistence context (and hence the database).

- The CRUD operations
  - em.persist(object) // Note: After this call object has id
  - em.find(… primaryKey)
  - em.merge(object) // Update and more …
  - em.remove(object)
  - …

    - Also: Possible to construct queries using em.createQuery(...) , more to come...

Automatic PC propagation

- Only for container managed EntityManagers
- If automatic PC propagation all EJBs involved use the same PC (EJBs calling EJBs)
    - If no propagation have to pass EM around (extra method parameter needed)
    - ... many, many, details...
- Other benefit of using EJB/Container



Entity class instance (@Entity) may be

- New, no persistent identity
- Managed, has identity in Persistence context, will be synchronized with real database at next commit
    - Commit automatically inserted
- Detached (not managed), has identity but not in PC will not be synchronized with database
- Removed, has identity in PC, will be removed from real database at next commit

Entities detached at

- Transaction commit (after last line in method, upcoming ...)
- Transaction rollback (exception)
- Explicitly detaching the entity

```
em.flush()  // Must do before...
em.detach(p) // ...this
```

- Clearing the PC: em.clear()
- Closing the EM: em.close()
- Passing by value (serializing)

Refresh

```
// State refreshed from database
em.refresh(p); // p managed (before and after)
```

Merge will create new managed copy or state copied

```
// p detached
p1 = em.merge(p)    // p1 != p !!! New object
p = em.merge(p)     // Could look like this, tricky...

// Also possible, skip returntype
em.merge(p)  // Any modification on p inside transaction persisted
```

Possible to check if managed

- em.contains(p)
  - True if p retrieved with em.find(), em.getReference(), em.persist(p) called or persist cascaded (upcoming)

Life cycle callback methods

- @PrePersist, @PostPersist,... (only one objects involved)
  - Bean validation triggered just before @PrePersist, @PreUpdate
- Possible customization
  - Separate listener class (possibly many objects involved)

Life cycle Ex 1

```
@Stateless
public class ShoppingCartBean
    @PersistenceContext
    EntityManager em;

    public OrderLine createOrderLine( Product p, Order o ){
        OrderLine ol = new OrderLine(p,o);
        em.persist( ol) ;
        return ol;
    }
}
```

This is container managed EM with transaction scoped PC

New Entity

PC active

Managed Entity (has id after call)

Detached Entity

Life cycle Ex 2

```
@Stateless
public class ShoppingCartBean
    @PersistenceContext
    EntityManager em

    public OrderLine createOrderLine( Product p, Order o ){
        OrderLine ol = new OrderLine(p,o);
        em.persist(ol) ;
        return ol;
    }

    public OrderLine updateOrderLine( OrderLine ol ){
        OrderLine ol2 = em.merge( ol) ;
        ...
        return ol2;
    }
}
```

PC Active

Detached

PC Active

Managed

Life cycle Ex 3

```
@Stateless
public class ShoppingCartBean
    @PersistenceContext
    EntityManager em

    public OrderLine createOrderLine( Product p, Order o ){
        OrderLine ol = new OrderLine(p,o);
        em.persist( ol) ;
        OrderLine ol2 = em.find(OrderLine.class, ol.getId());
        // ol == ol2 is TRUE
        return ol;
    }
}
```

PC Active

Retrieval returns same instance

Cascade, Fetching and OrphanRemoval

Cascade and Fetch

```
public class Publication ... {

    // No cascade on remove!
    @ManyToOne(cascade = {CascadeType.PERSIST,
        CascadeType.MERGE}, fetch = FetchType.LAZY)
    private Book book;
    // No cascade on remove!
    @ManyToOne(cascade = {CascadeType.PERSIST,
        CascadeType.MERGE}, fetch = FetchType.LAZY)
    private Author author;
    ...
}
```

OrphanRemoval

```
@OneToMany(orphanRemoval = true)
private Set<AppUser> members = new HashSet<>();
```

Possible to specify cascade and fetching behaviour using annotations on entity classes

Cascading

- Should storing, deleting, etc. apply to associated objects?
  - If car deleted probably engine should be deleted (also in database)! A cascade will automate the process, prefer!
  - If no cascade: Have to persist each single objects in correct order (avoid)
- Cascade types: ALL, PERSIST, REMOVE, MERGE, REFRESH, CLEAR, ALL
- Cascade may require bidirectional mappings

Fetching

- When to load associated objects
  - EAGER, when owner loaded
  - LAZY, when code executed
- Default (otherwise annotate)
  - @OneToOne, EAGER
  - @ManyToOne, EAGER
  - @OneToMany, LAZY
  - @ManyToMany, LAZY

Orphan removal

- Only for @OneToOne and @OneToMany
- If object removed from collection will be removed from database.

Persistence and Testing

Complicated to test persistence layer

- Would like to run tests outside container
- It's Possible to use an embedded container, see below
- … or using a Servlet (running in a container, avoid) …

Arquillian (tutorial)

- A testing platform for testing Java enterprise applications by enabling creation of integration, functional, behaviour tests …
  - Think: JUnit for JEE
- Possible to use embedded database (we don't).
  - JavaBD must be running

Transactions in Arquillian

```
@Inject
UserTransaction utx;

...
utx.begin();    // Start transaction
pubList.create(new Publication(
      new Book("aaa", 1f, "abc"), a));
pubList.create(new Publication(
      new Book("bbb", 2f, "abc"), a));
...
utx.commit();  // End transaction
...
```

In Arquillian tests must handle transactions explicit!

- Test classes are NOT EJBs
- Inject UserTransaction