6. rész: EJB-k tervezése és implementálása

Bakay Árpád
NETvisor kft
(30) 385 1711
arpad.bakay@netvisor.hu

A tananyag készült az
ELTE-IKKK
projekt támogatásával

Tartalom

- Session EJB-k - folyt
- Tranzakciók és biztonság
- Entity EJB-k

A Session EJB-k
-- „ügyintéző ablakok” hasonlat

- Stateless:
  - Minden szükséges irat mindig nálam van, tehát bármelyik ablakhoz mehetek, sőt, több különböző helyszínen is intézhetem a dolgaimat
- Stateful:
  - Otthagyom a iratokat, de ha rövid időn belül visszamegyek, még ugyanannál az ablaknál megtalálom a papírokat.
  - Ha később megyek, akkor már akármelyik kisasszonyhoz mehetek, ő lesöpri az asztalát, és hátulról előhozza a dossziémat.
  - DE: Ha leég az az iroda, kezdhetem előlről egy másik helyszínen!
    - Kivéve, ha elektronikus papírkezelés lenne… ez a session failover

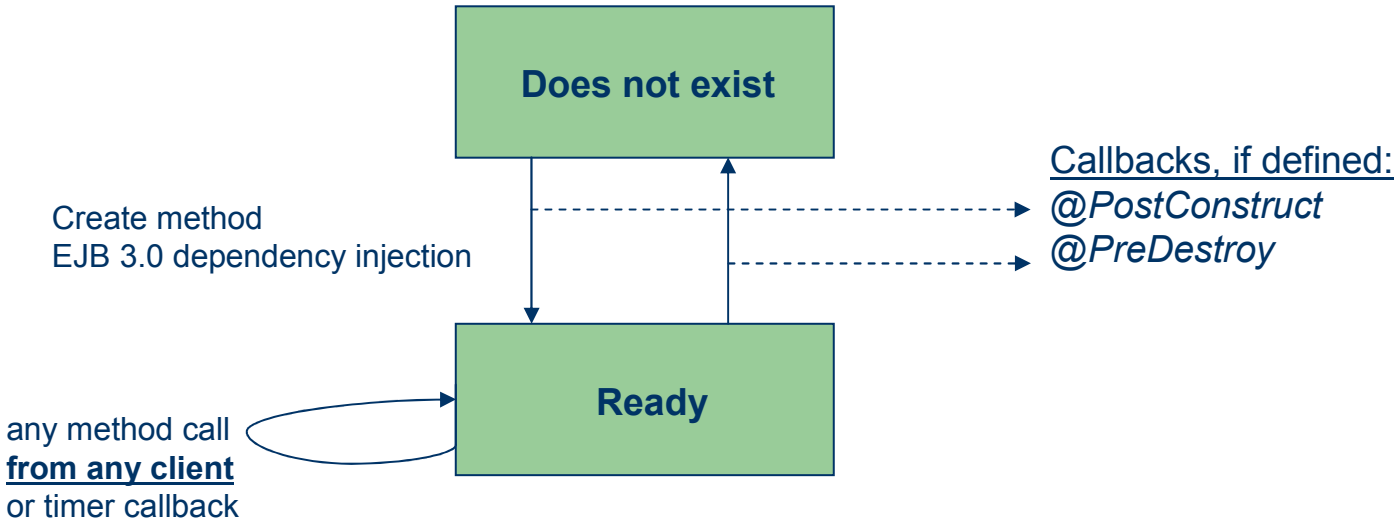
Session EJB általános vonások

- A Session EJB a kliens „távoli megbízottja”
- Egy bean handle csak egy klienshez tartozik
  - De lehet hogy több klienst szolgál ki egy vagy több valódi bean instance
- Bármely bean remote (RMI) és/vagy lokális interfészekkel készülhet el.

Stateless Session bean

- Nincsen *klienshez kötött* lokális állapota
- Életciklusáról a container gondoskodik
  - Pl. a kérések gyakorisága alapján növeli vagy csökkenti a bean-ek számát.
- Multiplexálás egyszerű
  - A leghamarabb felszabaduló instance szolgálja ki
  - Akár több több között is load balance-olhatunk ha a JNDI intelligensen szolgálja ki a lookup-okat
- Gyors, mert nem kell state adatokat előkotorni
- *VISZONT*: minden szükséges adatot át kell adni paraméterként minden hívásnál

Stateless EJB Életciklusa



Stateless bean – amit a kliens lát

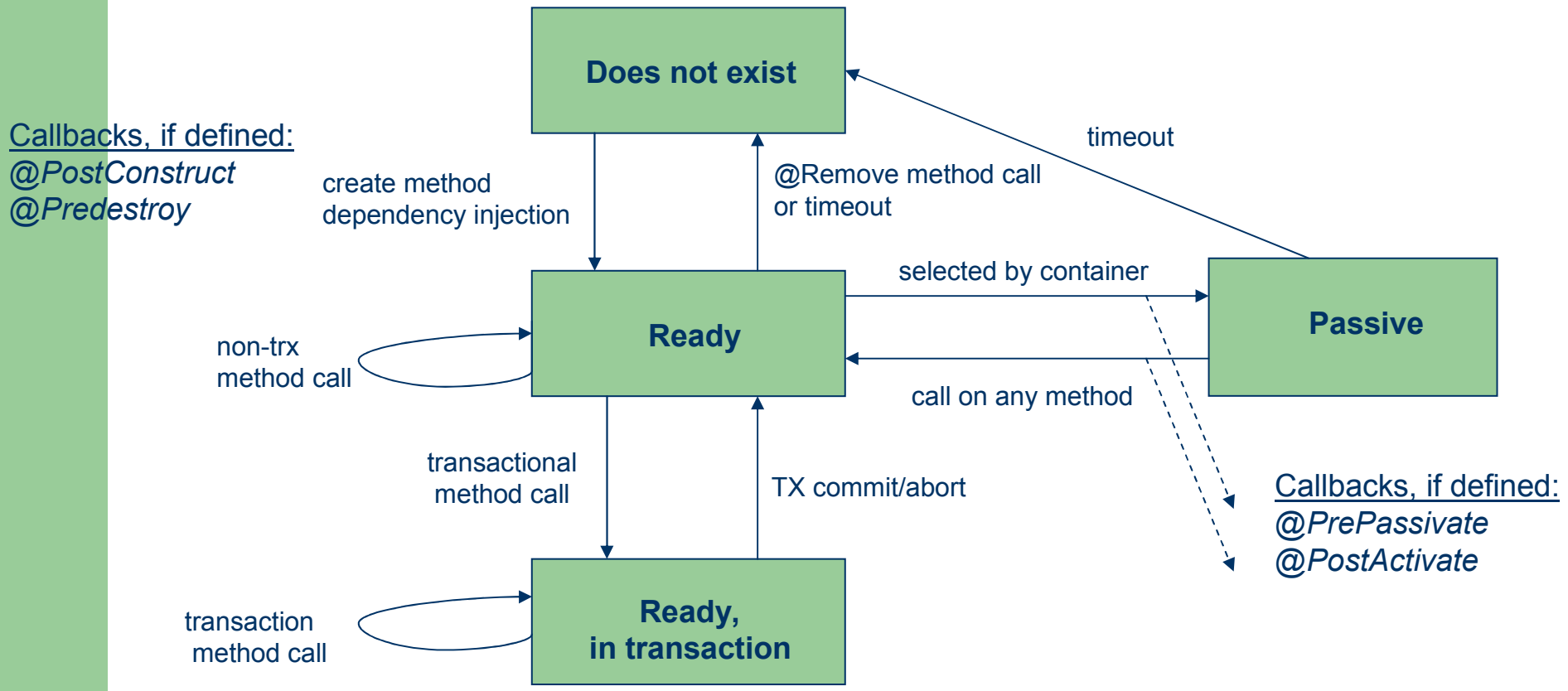
- Példa EJB: Szotar.class
- Kliens kód:

```
class SzotarClient {
 @EJB Szotar sz;   // dependency injection
 void fordit(void) {
   System.out.println(sz.angolMagyar("experience"));
   System.out.println(sz.magyarAngol("botrányos"));
   // lehet, hogy másodjára 'sz' már egy másik bean instance
   volt!
 }
}
```

Stateful Bean - különbségek

- Míg az SL-EJB esetében a referencia mögött az EJB instance gond nélkül cserélhető…
  - (így történik a pooling)
- … itt már a containernek gondoskodni kell az állapot áttelepítéséről is:
  - save to cache
    - amikor felszabadítja az eddig használt instance-et
  - restore from cache
    - amikor újra hívja a kliens a kirakott bean-t

Stateful EJB életciklusa



Stateful bean client view

- Példa EJB: Calculator.class
- Kliens kód:

```
class CalcClient {
 @EJB Calculator calc;   // dependency injection
 void kiszamol(void) {
        calc.turnOn()
        calc.add(5);
        calc.add(12);
        calc.multiply(3);
        System.out.println(calc.read());  // 51
        calc.turnOff();  // @remove annotated method
                // illik meghívni, különben a timeoutra kell várni
 }
}
```

EJB tranzakciók és biztonság

Tranzakciók kívánatos tulajdonságai „ACID”

- Atomicitás
  - Mindet vagy semmit végrehajt
- Consistency
  - Az adatbázis a művelet után a constrainteknek megfelel
- Isolation
  - Párhuzamos tranzakciók nem érzékelik egymást
- Durability
  - A változások tényleg maradandók

Tranzakciós kontextus

- Egy, a tranzakciót azonosító token, amely a hívások láncolátával párhuzamosan az ú.n. EJBContext-ben terjed
  - Egészen a klienstől indulhat
- Az egyes metódusok jelezhetik, hogy milyen tranzakciós környezetben szeretnek futni
  - @TransactionAttribute(xxx) method annotation
    - MANDATORY - kell, hogy már legyen tx
    - REQUIRED - csinál, ha eddig nem volt
    - REQUIRES_NEW - mindenképpen újat csinál
    - SUPPORTS - ha van tx, használja
    - NOT_SUPPORTED - nem használja
    - NEVER - csak tx nélkül szabad hívni

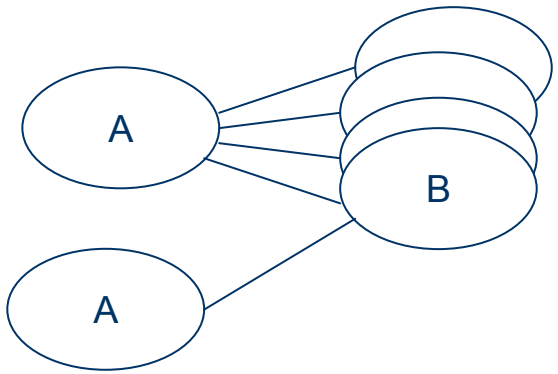
Biztonsági kontextus

- A műveletet kezdeményező user azonosítója (principal)
  - a tranzakciós kontextushoz hasonlóan, az EJBContext-ben terjed
- Használata:
  - Programból: EJBContext objektumon
    - EJBContext.getCallerPrincipal()
    - EJBContext.isCallerInRole („adminisztrátor”)
  - Deklaratíven: metadata annotation
    - @DeclareRoles(“payroll”) @Stateful class FizetesBean { }

3. EJB Entity-k

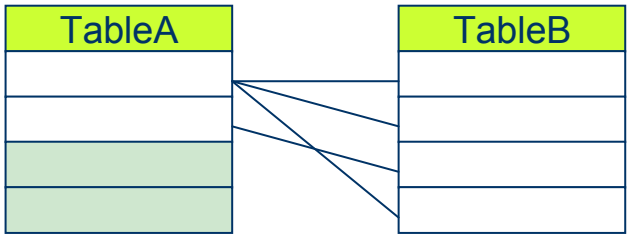
Objektum-relációs leképezés

OO



A szoftvertervezés és fejlesztés bevált szemlélete

RDBMS



Az adattárolás legmegbízhatóbb skálázhatóbb módja

EJB Entity-k

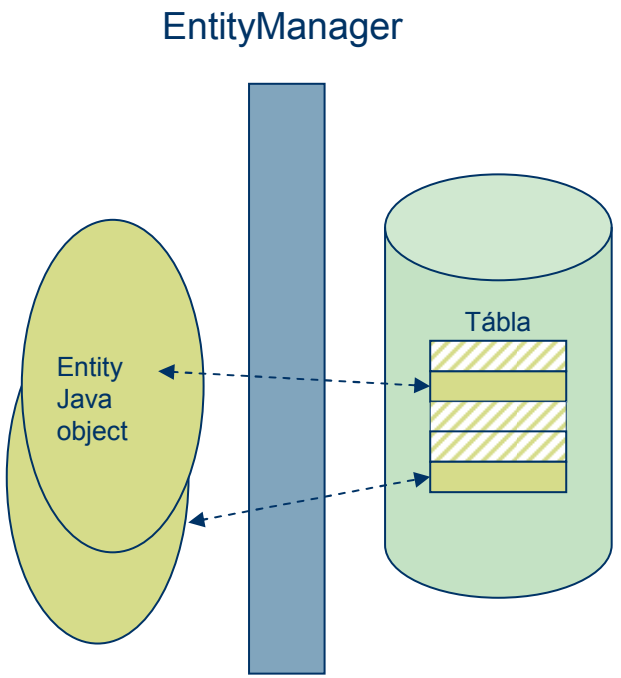
- Elsődleges céljuk az objekt-relációs leképezés
- Kevés (vagy inkább semmi) üzleti logikát tartalmaznak
- Jellemzően minden column-ra egy property
  - A PK-t @ID jelzi
    - Lehet automatikusan generált: @ID(generate=AUTO)
  - A többi oszlop attr @basic - de ez a default
  - Nem illeszkedő nevek: @column annotation
    - @column(name = „TEL_NO”) getTelephone()

Entity Bean: relációk

- 4 típus:
  - OneToOne
  - OneToMany
  - ManyToOne -
  - ManyToMany -- kapcsolótábla kell a DB-ben
- Adattípus *Many estén: valamilyen collection
  - List, Set, Collection
    - bármelyiket tudja kezelni, generálásnál választható
- Bidirectional relations
  - Oda és vissza irány
    - az FK->PK irány alapján!!!
  - A „visszafelé” oldalon a „mappedBy” attribútum jelzi, hogy melyik az oda-irány

Az EntityManager

- Az EJB 2.0 az Entity életciklusa az Entity-nek küldött üzeneteken keresztül változott.
- Az EJB 3.0-ben új EM szolgáltatás használata egyszerűbb és futásban hatékonyabb
- Az EM egy intelligens replikátor, ami vezérli az O/R mappinget,
  - Egyetlen EM egyszerre több különböző típusra, ill. példányra is



EM folyt

- Leggyakrabban a Session bean minden metódus elején nyit egy „tiszta” EM-et, és az összes objektumot ezzel kezeli, majd a hívás végén „öblít” az EM (ez a default, „transaction scoped EM”)
- EM kreálása a 3.0-ban:

  „dependency injection”

```
@PersistenceContext EntityManager em;
```

Kitérő: Dependency Injection

Az EJB 3.0 új, automatikus inicializálási technikája

- annotációk jelzik a konténernek, hogy hova mit kell injektálni, amikor egy példányt kreál
- JNDI-ből dolgozik
- a default akció jellemzően megfelel, de speciális igényekhez az annotációk paraméterezhetők

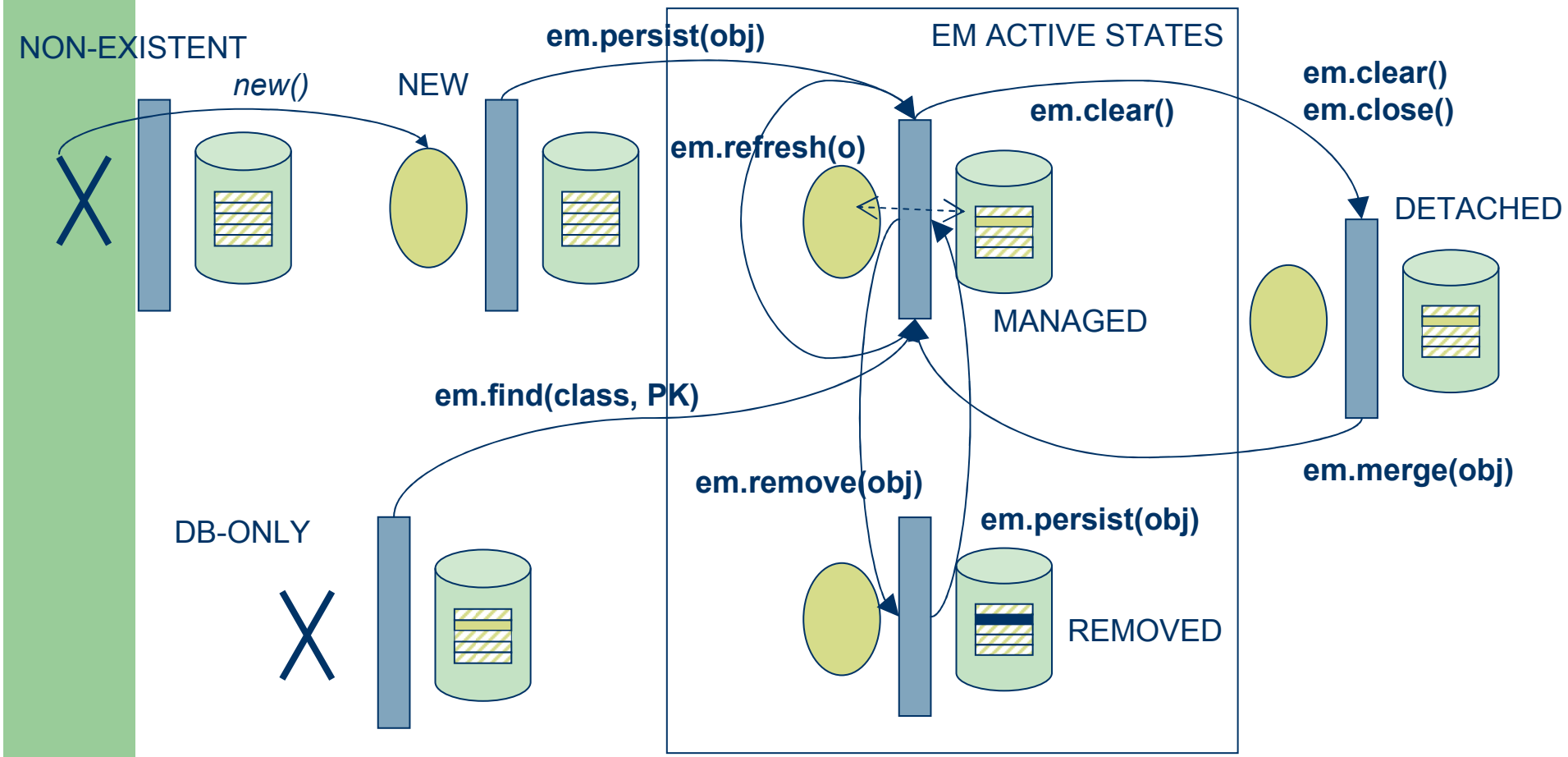
- példa (példányváltozó egy session EJB-ben):

```
@PersistenceContext EntityManager em;
```

– v.ö. A JNDI hagományos használata:

```
EntityManager em;
public void initEM(SessionContext ctx) {
   em = (EntityManager)ctx.lookup('MyEM');
}
```

Az Entity életciklusa



Megjegyzések az életciklus ábrához

- A MANAGED és REMOVED entity-k DB képe csak em.flush(), ill. transaction commit után garantált.
- A leggyakoribb u.n. transaction scoped EM-eknél a tr. végeztével az EM bezárul, és minden entity DETACHED lesz.
    - Az Extended EM-nél maradnak bent.
- A relációk követése a CASCADE= annotation értéke szerint
    - ALL, PERSIST, MERGE, REFRESH, REMOVE

4. Egyebek

EJB verziók
és más komponens technológiák

- Az EJB 2.1 „nehézsúlyú” koponensek technológiája.
  - A Bean class-ok kötelezően leszármazotjai egy, a contianer által biztosított osztálynak
  - Az Entity perzisztenicia egyszerűen használható, de lassú
  - Sok repetitív kódolás, pl. bean - remote interfész - local interfész
- EJB 3.0: forradalmi változtatás, jelentős könnyítések
  - Bean-ek POJO-k, és akár container nélkül is használhatók/tesztelhetők
  - Megváltozott perzisztencia technológia, kevesebb kód, stb.
- Érdekes egyéb containerek Spring Framework, PicoContainer, stb.
  - Még könnyebb súlyúak
  - Kevésbé elterjedtek és csak 1-1 szoftver támogatja

Köszönöm a figyelmet!