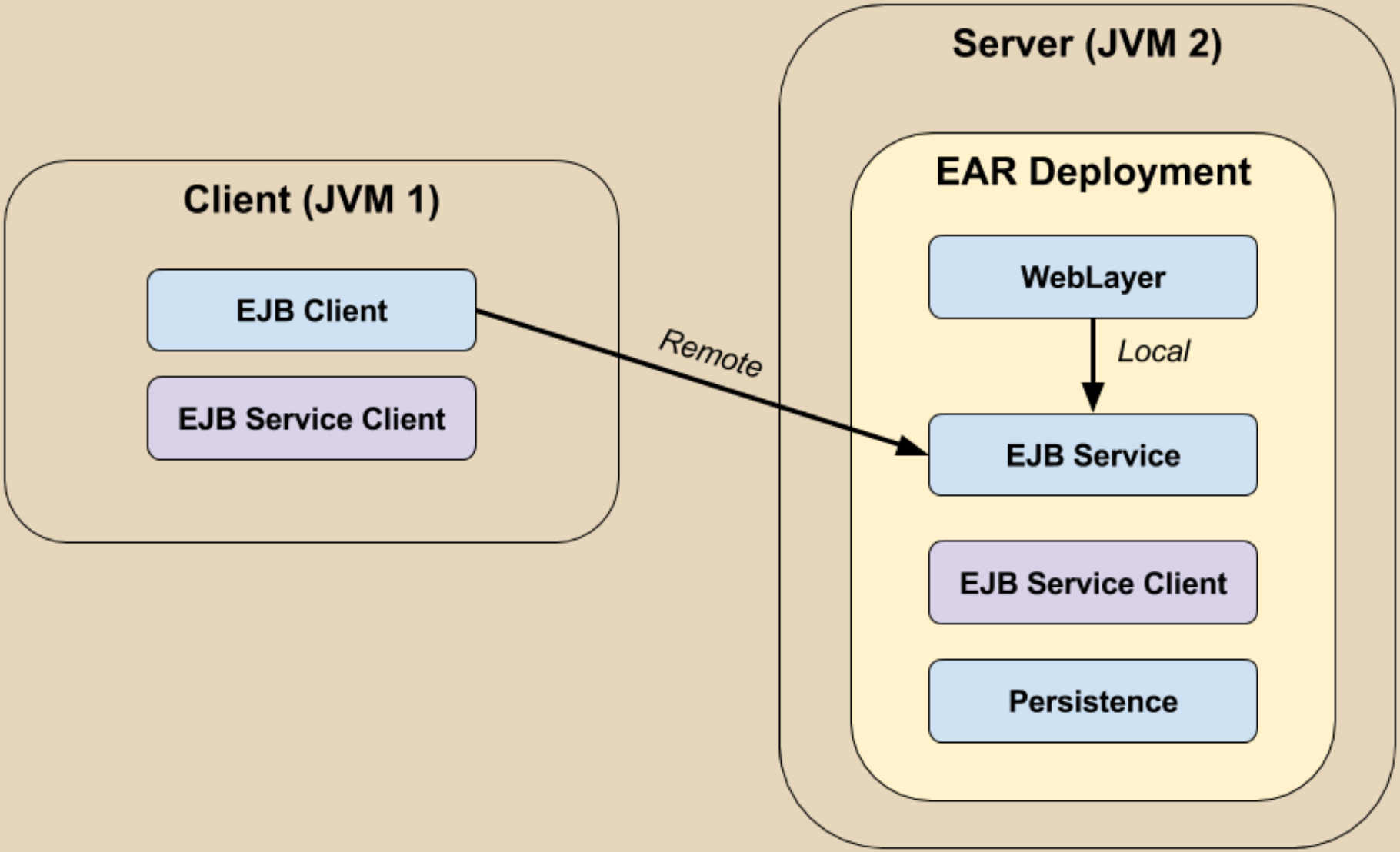
# DiskStore #maven

Remote EJB, JNDI, Dependency management, Service client, Context root, InitialContext, MyBatis 3

**Óbudai Egyetem**, Java Enterprise Edition
Műszaki Informatika szak
Labor 4

Bedők Dávid
2018-01-17
**v1.1**

# Local vs Remote EJB



# Előkövetelmények

Mire van szükség ahhoz hogy egy EJB service-t távolról meghívjunk?

- Ismernünk kell a kiszolgáló *server* futásának helyét (gép, port)
- Ismernünk kell a *server* típusát, esetleg konfigurációs körülményeit (application server, jndi rules, ...)
- Ismernünk kell a *enterprise* alkalmazás meta/descriptor adatait (application name, ejb module name, ...)
- Ismernünk kell a meghívandó *service* aláírását, és rendelkeznünk kell azokkal az osztályokkal, melyek mindehhez szükségesek (szerializálható paraméterek, visszatérési értékek és kivételek, illetve mindezek által hivatkozott (fordításukhoz szükséges) típusok)

A forráskódot érintő függőségek (**EJB Service Client**) miatt a Remote EJB használata legtöbbször olyan fejlesztési igényeknél jelentkezik, ahol a kliens és a szerver oldali fejlesztés is egy kézben van (a kliens kódja is a termék része). Ahhoz hogy ezen függőségek redundancia mentesen legyenek megvalósítva, az enterpise alkalmazás **csomagolásának átszervezésére** van szükség.

# JNDI

## Java Naming and Directory Interface

- A JNDI segítségével elosztott alkalmazások absztrakt, erőforrás-független módon képesek szolgáltatásokat vagy regisztrált erőforrásokat keresni és kikérni.
- Objektumok halmazát tartja karban, tipikusan "könyvtár" (fa) strukúrában
- Biztosítja az elemek regisztrációját, keresését, lekérését
- Képes értesíteni a klienst egy elem megváltozásáról (eseményküldés)
- A kulcs tipikusan `String`, de egyébként egy `Name` interface-t implementáló példány
- Használat tipikus esete:
  - Konfigurációs paraméterek beállítása/megadása (`jndi.properties` vagy egy `Hashtable<String, String>` feltöltése)
    - Meg kell adni a szerver `host` nevét, `port`-ját, a kommunikáció `protocol`-ját
    - Meg kell adni az `InitialContextFactory` osztály *full qualified* nevét (az osztálynak a *classpath*-on kell lennie)
    - kiegészítő pl. authentikációs adatok megadása
  - `InitialContext` létrehozása szerver specifikus factory-n keresztül
  - `javax.naming.Context` példány `lookup( [JNDI name] )` metódusának segítségével a keresett erőforrás kikérése

# JNDI használata

**Remote**: a konfigurációról gondoskodni szükséges (lehet `jndi.properties` állományt is használni és a *classpath*-ra elhelyezni).

```
1 Hashtable<String, String> jndiProperties = new Hashtable<>();
2 jndiProperties.put("java.naming.factory.initial",
      "org.jboss.naming.remote.client.InitialContextFactory");
3 jndiProperties.put("java.naming.provider.url",
      "remote://localhost:4447");
4 Context context = new InitialContext(jndiProperties);
5 context.lookup("...");
```

Az itt megadott értékek JBoss 6.4 specifikusak, a hivatkozott `InitialContextFactory` osztálynak a *classpath*-on kell lennie. A `String` kulcsok helyett lehet használni az alábbi konstansokat is:
`javax.naming.Context.INITIAL_CONTEXT_FACTORY`
`javax.naming.Context.PROVIDER_URL`

**Local**: a konfigurációt biztosítja a container (a beállított `jndi.properties` állomány már a *classpath*-on van).

```
1 Context context = new InitialContext();
2 context.lookup("...");
```

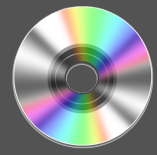
## JNDI név

[context]/[application-name]/[module-name]/[bean-name]![full-qualified-interface-name]

- **context**
  - JavaEE standard
    - `java:comp`: az adott komponens számára érhető el (*ejb*-n belül)
    - `java:module`: az adott modul számára érhető el (*ejb module*-on belül)
    - `java:app`: az adott alkalmazás számára érhető el (*ear*-on belül)
    - `java:global`: az adott alkalmazás szerver domain számára érhető el (*standalone domain*-en belül)
  - JBoss specifikus
    - `java:jboss/exported`: container-en kívülről érhető el
- **application-name**
  - Az *ear deployment descriptor*án (`application.xml`) keresztül állítható be.
- **module-name**
  - Az *EJB module deployment descriptor*én (`ejb-jar.xml`) keresztül állítható be.
- **bean-name**
  - A `@Stateless( name = "[BEAN_NAME]")` annotáció `name` értékén keresztül állítható be.

# Követelmények technikai részletei

- A szerver oldali üzleti komponenst kiszolgáló alkalmazás szerver ún. **kliens könyvtára**, mely elsősorban az adott verziójú alkalmazás szerverrel való kommunikáció végett fontos (protokollok, szerver specifikus konfigurációs beállítások, ...) (**JBoss client library / WebLogic (full)client jar**).
- Az alkalmazott alkalmazás szerver által használt **EJB implementáció** (a kliens oldal *enterprise container* nélkül fog futni, így a JavaEE 6.0 API nem lesz elegendő, helyette ezen (teljes) API JBoss/WebLogic implementációját fogjuk a kliens oldalon a *classpath*-ra elhelyezni).
- A Remote kommunikációban részt vevő típusok példányai a hálózaton keresztül fognak közlekedni, így ezek **szerializálhatósága/deszerializálhatósága** elengedhetetlen lesz, erről programozottan gondoskodni szükséges (Java-ban ehhez legtöbbször elegendő a `Serializable` interface-t implementálni, és esetleg a nem szerializálható mezők esetén a `transient` kulcsszót használni).

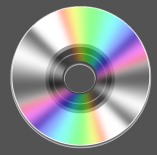


**Feladat**: azonos módon a "BookStore"-hoz hozzunk létre egy **lemezboltot**. EJB service-eken keresztül valósítsuk meg a lemezekre nézve a CRUD műveleteket.

- Az adatbázis schema felépítése teljesen azonos lehet a "BookStore" projektben megismerttel.
- A webfelület kialakítása hasonló lehet a "BookStore" projektben megvalósított felülettel.
- A perzisztens réteg ne JPA-t használjon ORM-ként, hanem egy egyszerűbb, natív lekérdezésekre épülő (de *type-safe*) megoldást mutasson be a **MyBatis 3** 3rd party library segítségével.
- A lemez egyedi azonosítójának (`reference`) ismeretében lehessen lekéri a lemez adatait távolról is (**remote ejb call**).

# Project stuktúra

## Alprojektek, modulok



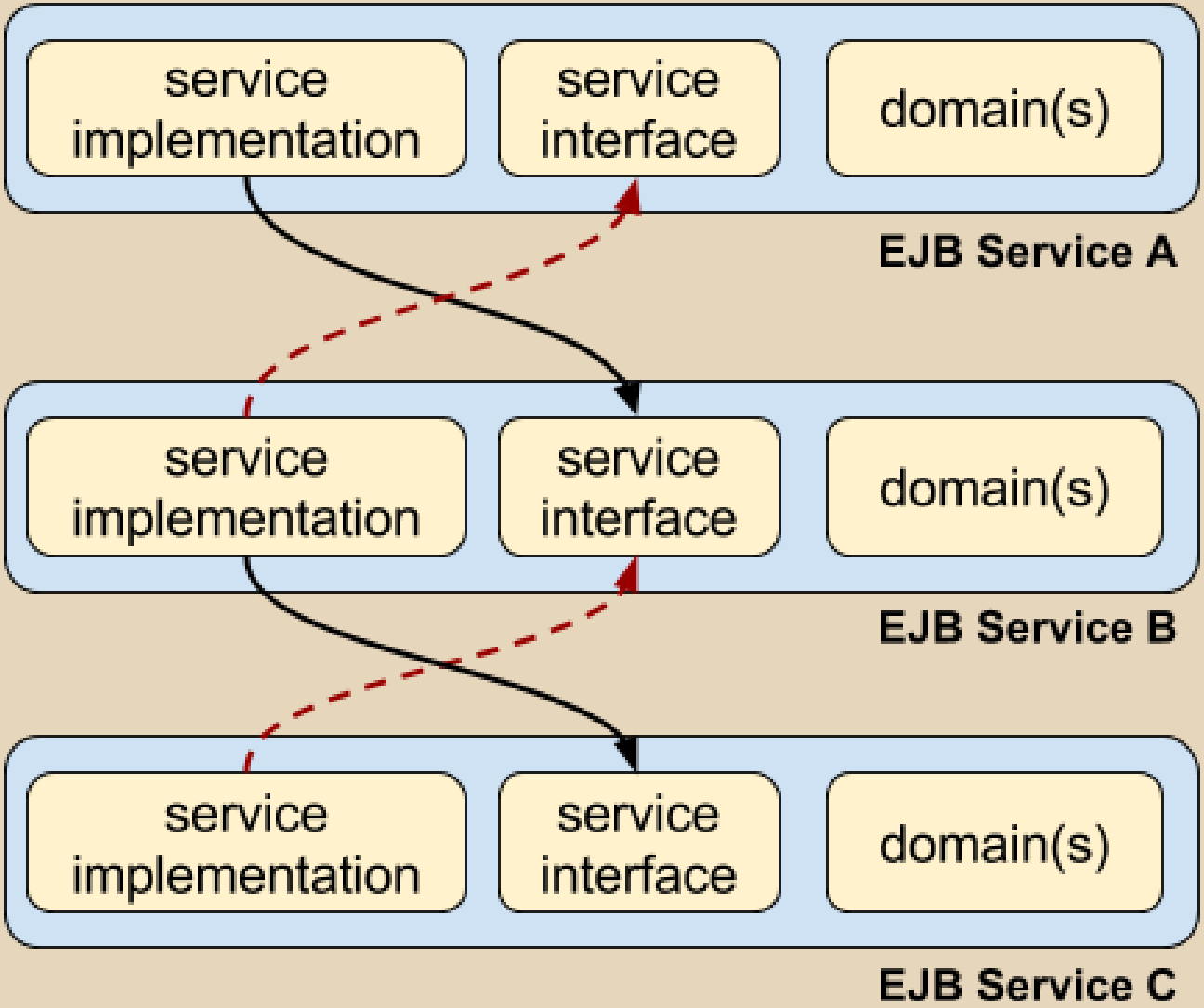
- **diskstore** (root project)
  - **ds-weblayer** (EAR web module)
    - `DiskPingServlet`
    - MVC minta szerint Servlet és JSP alapokon megvalósított webfelület
  - **ds-ejbservice** (EAR ejb module)
    - Üzleti metódusok implementációja
    - Local EJB interface
  - **ds-persistence** (EAR ejb module)
    - ORM réteg (MyBatis 3)
  - **ds-ejbserviceclient** (EAR library)
    - Szerializálható domain osztályok (stub-ok), kivételek
    - Remote EJB interface
  - **ds-client** (EJB client alkalmazás)

Része az EAR-nak: project + project

Része az EJB client alkalmazásnak: project + project

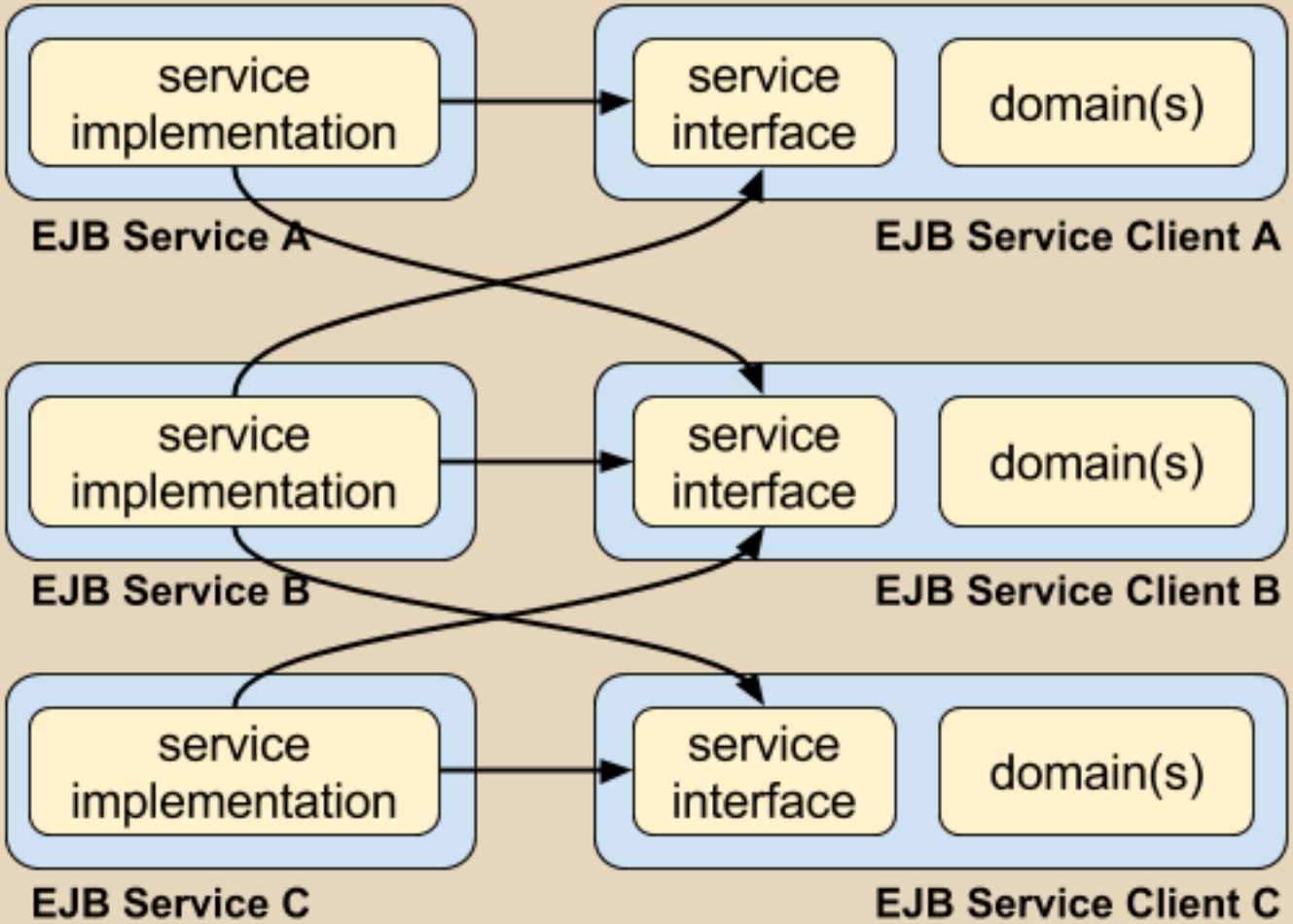
# EJB Modulok közötti kör függőség

## Nem működő megoldás



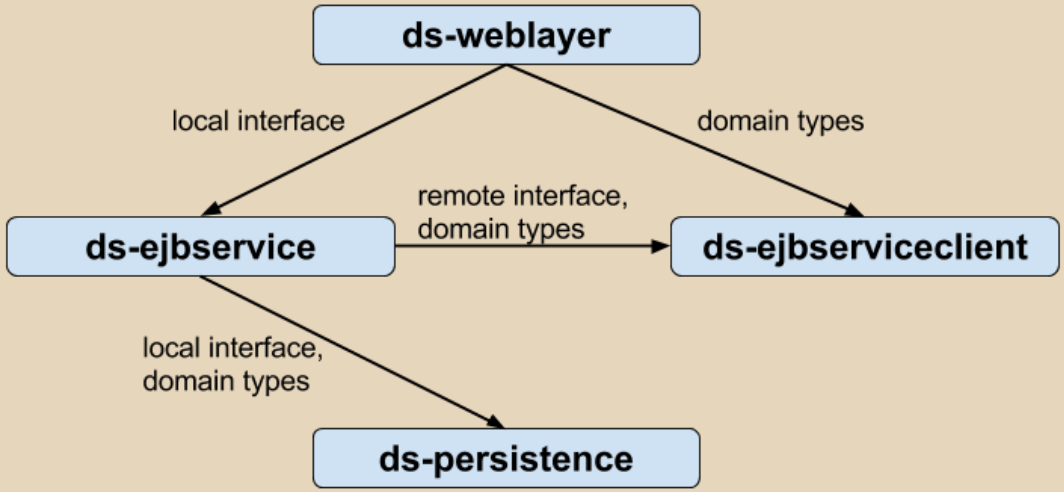
# EJB Modulok közötti kör függőség

## Működő megoldás (körök feloldása)



# Modulok közötti függőségek

## DiskStore project




A `ds-ejbserviceclient` kiszervezése a `ds-ejbservice` projektből jelenleg kizárólag a távoli EJB hívás végett történt. Általános esetben a módszer alkalmas a függőségi körök elkerülésére is, de most ez itt nem merült fel (gyakorlati hibát nem okozna az, ha a kliens classpath-ára feltennénk a teljes `ds-ejbservice` projekt artifactját, azonban ezzel súlyos elvi hibát okoznánk). Mivel a `ds-ejbservice` module-ban megmaradnak `@Local` EJB-k (lokálisan a teljes CRUD műveletet végre tudjuk hajtani, míg távolról csak egy lekérdezései lehetőségünk lesz), a `ds-weblayer` projekt függni fog a `ds-ejbservice` projekttől is a `ds-ejbserviceclient` mellett. A `ds-ejbservice` üzleti műveleteinek implementációja használni fogja a perzisztens réteget, csakúgy mint a *BookStore* példájában, így e téren a függőségi lánc a már megismerttel azonos.

# Adatbázis schema

## Lemez és kategóriája

```
1  CREATE TABLE diskcategory (
2    diskcategory_id INTEGER NOT NULL,
3    diskcategory_name CHARACTER VARYING(100) NOT NULL,
4    CONSTRAINT PK_DISKCATEGORY_ID PRIMARY KEY (diskcategory_id)
5  );
6
7  CREATE TABLE disk (
8    disk_id SERIAL NOT NULL,
9    disk_reference CHARACTER VARYING(100) NOT NULL,
10   disk_author CHARACTER VARYING(100) NOT NULL,
11   disk_title CHARACTER VARYING(100) NOT NULL,
12   disk_diskcategory_id INTEGER NOT NULL,
13   disk_price REAL NOT NULL,
14   disk_number_of_songs INTEGER NOT NULL,
15   CONSTRAINT PK_DISK_ID PRIMARY KEY (disk_id),
16   CONSTRAINT FK_DISK_DISKCATEGORY FOREIGN KEY (disk_diskcategory_id)
17     REFERENCES diskcategory (diskcategory_id) MATCH SIMPLE ON UPDATE RESTRICT ON DELETE
           RESTRICT
18 );
```

create-schema.sql

# EJB Service client

## A szerver és a kliens oldal számára is szükséges típusok

- Domain osztályok
  - `DiskStub`
  - `DiskCategoryStub`
    - Mivel az `Enum` ősosztály `implements Serializable`, semmi kiemelt figyelmet nem érdemel ez az osztály.
- Kivétel osztály
  - `ServiceException`
    - Mivel az `Exception` ősosztály `implements Serializable`, semmi kiemelt figyelmet nem érdemel ez az osztály.
- Remote interface
  - `DiskFacadeRemote`

# EJB Service client

## Build script

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/maven-v4_0_0.xsd">
3
4    <modelVersion>4.0.0</modelVersion>
5    <artifactId>ds-ejbserviceclient</artifactId>
6    <packaging>jar</packaging>
7    <name>DiskStore EJB Services Client</name>
8
9    <parent>
10     <groupId>hu.qwaevisz.diskstore</groupId>
11     <artifactId>diskstore</artifactId>
12     <version>1.0</version>
13   </parent>
14
15   <dependencies>
16     <dependency>
17       <groupId>org.jboss.spec</groupId>
18       <artifactId>jboss-javaee-6.0</artifactId>
19       <type>pom</type>
20     </dependency>
21   </dependencies>
22
23 </project>
```

*Compile time* csupán a `@Remote` annotáció miatt szükséges egy Java EE 6.0 API vagy implementáció, *Run time* azonban az EJB specifikáció további része is betöltésre kerül az **EJB kliens** oldaláról!

pom.xml

# Remote interface

## Disk adatainak lekérdezése egyedi azonosító alapján

```java
package hu.qwaevisz.diskstore.ejbserviceclient;

import javax.ejb.Remote;

import hu.qwaevisz.diskstore.ejbserviceclient.domain.DiskStub;
import hu.qwaevisz.diskstore.ejbserviceclient.exception.ServiceException;

@Remote
public interface DiskFacadeRemote {

  String BEAN_NAME = "DiskStoreService";

  DiskStub getDisk(String reference) throws ServiceException;

}
```

A `BEAN_NAME` egy publikus konstans, mely része lesz az EJB service JNDI nevének. Mivel a Remote interface publikus, szokás ezt az értéket itt tárolni.

`DiskFacadeRemote.java`

A `@Local` interface-hez képest a lényegi különbség a `@Remote` annotáció használata. Egy interface-en jelen lehet egyszerre mindkét annotáció, arra azonban figyeljünk hogy ez esetben minden későbbi változásnál (új *business method* készítésénél) megvizsgálandó hogy az új funkció elérhető-e távolról is.

# Domain osztály

## Disk adatai

A *BookStore* példában megismert BookStub-hoz képest a különbség csupán a Serializable interface implementálása annak érdekében, hogy a típus példányai a hálózaton keresztül közlekedni tudjanak.

```
1  package hu.qwaevisz.diskstore.ejbserviceclient.domain;
2
3  import java.io.Serializable;
4
5  public class DiskStub implements Serializable {
6
7    private String reference;
8    private String author;
9    private String title;
10   private DiskCategoryStub category;
11   private double price;
12   private int numberOfSongs;
13
14   [..]
15 }
```

DiskStub.java

# EAR finomhangolása

Néhány apró ám annál fontosabb finomhangolási lehetőséget érdemes megvizsgálni az EAR összeállítása során.

- EAR 3rd party library-k használata (*Gradle* esetén `earlib dependency configuration`. *Maven* esetén `jar type dependency`) (melyek nem ejb/web module-jai az EAR-nak, azonban olyan erőforrások, melyeket az elindított server domain nem töltött be (mivel legtöbbször csak ezen EAR artifact számára fontosak)).
  - EJB Service client (nem **transitive** függőség!)
  - MyBatis 3
- EAR Java EE verziójának (`version`), megjelenítési (`displayName`) és hivatkozási nevének (`applicationName`) beállítása. Utóbbi része az EAR-on belüli szolgáltatások JNDI nevének (alapértelmezetten az `ear` fizikai file nevével egyezik meg).
- EAR-on belüli webalkalmazás(ok) **context root** tulajdonságának beállítása
  - A *context root* segítségével lehet elérni a webalkalmazások `Servlet` osztályait, kiemelten publikus adat. Értékét az `application.xml`-ben definiáljuk, alapértelmezés szerint legtöbbször az `ear`-on belüli `war` fizikai file neve (alkalmazott build rendszer függvénye).

# EAR Deployment descriptor

```xml
<?xml version="1.0"?>
<application xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/application_6.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    version="6">
  <application-name>diskstoreapp</application-name>
  <display-name>Disk Store Application</display-name>
  <module>
    <web>
      <web-uri>ds-weblayer.war</web-uri>
      <context-root>diskstore</context-ro
    </web>
  </module>
  <module>
    <ejb>ds-persistence.jar</ejb>
  </module>
  <module>
    <ejb>ds-ejbservice.jar</ejb>
  </module>
  <library-directory>library</library-directory>
</application>
```

*Gradle* esetén a projectek verziószáma *empty String*, míg *Maven* esetén tipikusan jelöljük pl. `1.0`-ként. Utóbbi esetben a `jar` file nevek tartalmazni fogják a verziószámot.

Legyen az alkalmazás hivatkozási neve **diskstoreapp**, míg a ds-weblayer webmodule *context root*-ja **diskstore**. A 3rd party library-kat a **library** alkönyvtárba helyezzük el.

application.xml

# EAR finomhangolása - I

## ds-ear build script

```
1  <project [..]>
2    [..]
3    <dependencies>
4      [..]
5      <dependency>
6        <groupId>hu.qwaevisz.diskstore</groupId>
7        <artifactId>ds-ejbserviceclient</artifactId>
8        <version>${project.parent.version}</version>
9        <type>jar</type>
10       <exclusions>
11         <exclusion>
12           <groupId>org.jboss.spec</groupId>
13           <artifactId>jboss-javaee-6.0</artifactId>
14         </exclusion>
15       </exclusions>
16     </dependency>
17   </dependencies>
18   [..]
19 </project>
```

A <type> értéke jar, mely jelöli hogy az EAR számára ez csupán 3rd party library, nem *module*. A ds-persistence project *MyBatis 3* függősége viszont automatikusan bekerül az EAR library-jai közé.

A ds-ejbserviceclient nem szabad hogy **transitive** függőségként legyen része az EAR-nak, hiszen akkor a JBoss specifikus Java EE 6.0 API implementációját is tartalmazná a *deployment*. Ugyanezt az <exclusions> blockot szerepeltetni kell majd a ds-weblayer project esetén is (mivel a *Maven* a webapp lib könyvtárába is bemásolná a függőségeket).

pom.xml

# EAR finomhangolása - II

## ds-ear build script

```
1  <project [..]>
2    [..]
3    <build>
4      <directory>build</directory>
5      <finalName>${project.parent.artifactId}-${project.version}</finalName>
6      <plugins>
7        <plugin>
8          <groupId>org.apache.maven.plugins</groupId>
9          <artifactId>maven-ear-plugin</artifactId>
10         <version>${version.plugin.ear}</version>
11         <configuration>
12           <applicationName>diskstoreapp</applicationName>
13           <displayName>Disk Store Application</displayName>
14           <version>6</version>
15           <defaultLibBundleDir>library</defaultLibBundleDir>
16           <modules>
17             <webModule>
18               <groupId>hu.qwaevisz.diskstore</groupId>
19               <artifactId>ds-weblayer</artifactId>
20               <contextRoot>diskstore</contextRoot>
21             </webModule>
22           </modules>
23         </configuration>
24       </plugin>
25     </plugins>
26   </build>
27 </project>
```

A version alapértelmezett értéke 3, míg a libraryDirectory-é *root* könyvtár (nem ajánlott ezt alkalmazni).

A JNDI név részét képző applicationName értéke **diskstoreapp** lesz, míg a pontos artifact URI-val megcímzett ds-weblayer *context root*-ja **diskstore**.

pom.xml

# EJB module deployment descriptor

## ds-ejbservice subproject

src | main | resources | META-INF | ejb-jar.xml

```
1 <ejb-jar xmlns="http://xmlns.jcp.org/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
          http://xmlns.jcp.org/xml/ns/javaee/ejb-jar_3_2.xsd"
3     version="3.2">
4     <module-name>dsservicemodule</module-name>
5 </ejb-jar>
```

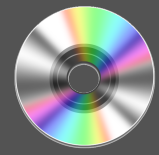
ejb-jar.xml

# Bean név konfiguráció

## ds-ejbservice subproject

```
1 package hu.qwaevisz.diskstore.ejbservice.facade;
2 [..]
3 @Stateless(mappedName = "ejb/diskFacade", name = DiskFacadeRemote.BEAN_NAME)
4 public class DiskFacadeImpl implements DiskFacade, DiskFacadeRemote {
5
6   [..]
7
8 }
```

DiskFacadeImpl.java



### JNDI név

[context]/[application-name]/[module-name]/[bean-name]![full-qualified-interface-name]

### DiskStore remote JNDI név

context: **java:jboss/exported/**
application-name: **diskstoreapp/**
module-name: **dsservicemodule/**
bean-name: **DiskStoreService!**
full-qualified-interface-name: **hu.qwaevisz.diskstore.ejbserviceclient.DiskFacadeRemote**

# Deployment

## diskstore.ear

[JBOSS_HOME] | standalone | log | server.log

```
18:08:08,665 INFO
[org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeploymentUnitProcessor] (MSC
service thread 1-6) JNDI bindings for session bean named DiskStoreService in deployment
unit subdeployment "ds-ejbservice.jar" of deployment "diskstore.ear" are as follows:

  java:global/diskstoreapp/dsservicemodule/DiskStoreService!hu.qwaevisz.diskstore.ejbservic
  java:app/dsservicemodule/DiskStoreService!hu.qwaevisz.diskstore.ejbservice.facade.DiskFac
  java:module/DiskStoreService!hu.qwaevisz.diskstore.ejbservice.facade.DiskFacade
  java:global/diskstoreapp/dsservicemodule/DiskStoreService!hu.qwaevisz.diskstore.ejbservic
  java:app/dsservicemodule/DiskStoreService!hu.qwaevisz.diskstore.ejbserviceclient.DiskFaca
  java:module/DiskStoreService!hu.qwaevisz.diskstore.ejbserviceclient.DiskFacadeRemote
  java:jboss/exported/diskstoreapp/dsservicemodule/DiskStoreService!hu.qwaevisz.diskstore.e
```

server.log

ds-weblayer tesztelése:

http://localhost:8080/diskstore/DiskPing

http://localhost:8080/diskstore/DiskList

# EJB Client

## Build script

```xml
<project [..]>
  [..]
  <dependencies>
    <dependency>
      <groupId>hu.qwaevisz.diskstore</groupId>
      <artifactId>ds-ejbserviceclient</artifactId>
      <version>${project.parent.version}</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>${version.log4j}</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>jboss</groupId>
      <artifactId>client</artifactId>
      <version>1.0</version>
      <scope>system</scope>
      <systemPath>${project.basedir}/lib/jb
    </dependency>
  </dependencies>
</project>
```

A JBoss specifikus kommunikáció (nevezéktan, protocol) felépítéséhez szükséges a **jboss-client.jar** megtalálható a `[JBOSS-HOME]\bin\client\jboss-client.jar` elérési úton. Ezen 3rd party elegánsabb módja lehet egy *JBoss repository*-ból való kikérése *artifact uri* alapján. Demonstrációs céllal most ezt egy helyi könyvtárból olvassuk be (az *EJB Client* project gyökerében található **lib** könyvtárból).

pom.xml

# EJB Client

## Pszeudokód

- `InitialContext` létrehozása
  - Programozott módon, `Hashtable<String, String>` feltöltésével vagy
  - *Classpath*-ra elhelyezett `jndi.properties` segítségével
  - Mindkét esetben szükség lesz egy JBoss specifikus kulcsra is:
    - **Kulcs**: `jboss.naming.client.ejb.context`
    - **Érték**: `true`
- DiskService (`DiskFacadeRemote` proxy) kikérése JNDI-ból
  - `context.lookup([JNDI name])` metódus segítségével
- Disk lekérése (`DiskStub` példány)
  - `diskFacadeRemote.getDisk([disk reference])` metódus segítségével

# Initial Context létrehozása

## Programozott megoldás

```
1  package hu.qwaevisz.diskstore.client.context;
2  [..]
3  public class ProgrammedContextFactory implements ContextFactory {
4
5    private static final String JBOSS_NAMING_CLIENT_EJB_CONTEXT_KEY
         = "jboss.naming.client.ejb.context";
6
7    @Override
8    public Context getContext() throws NamingException {
9      final Hashtable<String, String> jndiProperties = new
           Hashtable<String, String>();
10     jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY,
           "org.jboss.naming.remote.client.InitialContextFactory");
11     jndiProperties.put(Context.PROVIDER_URL,
           "remote://localhost:4447");
12     jndiProperties.put(JBOSS_NAMING_CLIENT_EJB_CONTEXT_KEY,
           "true");
13     return new InitialContext(jndiProperties);
14   }
15 }
```

ProgrammedContextFactory.java

# Initial Context létrehozása

## jndi.properties alapú konfiguráció

src | main | resources | jndi.properties

```
1 java.naming.factory.initial=org.jboss.naming.remote.client.InitialContextFactory
2 java.naming.provider.url=remote://localhost:4447
3 jboss.naming.client.ejb.context=true
```

jndi.properties

```
1 package hu.qwaevisz.diskstore.client.context;
2 [..]
3 public class JndiPropertiesContextFactory implements
      ContextFactory {
4
5   @Override
6   public Context getContext() throws NamingException {
7     return new InitialContext();
8   }
9 }
```

JndiPropertiesContextFactory.java

# JNDI lookup

```
1  package hu.qwaevisz.diskstore.client;
2  [..]
3  public class DiskClient {
4
5    private ContextFactory contextFactory;
6
7    public DiskClient(ContextFactory contextFactory) {
8      this.contextFactory = contextFactory;
9    }
10
11   public DiskStub getDisk(final String reference) {
12     DiskStub disk = null;
13     try {
14       final DiskFacadeRemote facade =
               this.getDiskService(this.contextFactory.getContext());
15       disk = facade.getDisk(reference);
16       LOGGER.info(disk);
17     } catch (final ServiceException e) {
18       LOGGER.error(e, e);
19     } catch (final NamingException e) {
20       LOGGER.error(e, e);
21     }
22     return disk;
23   }
24
25   private DiskFacadeRemote getDiskService(Context context) throws NamingException {
26     return (DiskFacadeRemote) context.lookup("[JNDI-NAME]");
27   }
28 }
```

diskstoreapp/dsservicemodule/DiskStoreService!
hu.qwaevisz.diskstore.ejbserviceclient.DiskFacadeRemote

# Kliens oldali naplózás

## Log4J

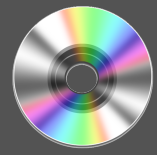
src | main | resources | log4j.xml

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
3  <log4j:configuration debug="true"
4    xmlns:log4j='http://jakarta.apache.org/log4j/'>
5
6    <appender name="console"
         class="org.apache.log4j.ConsoleAppender">
7        <layout class="org.apache.log4j.PatternLayout">
8      <param name="ConversionPattern"
9        value="%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n" />
10       </layout>
11   </appender>
12
13   <root>
14     <level value="DEBUG" />
15     <appender-ref ref="console" />
16   </root>
17
18 </log4j:configuration>
```

log4j.xml

# Kliens alkalmazás tesztelése

## Futtatás



Alkalmazás futtatása pl. az IDE által indított JVM-ből.

```
1 2017-11-08 16:48:53 DEBUG EJBClientContext:720 -
      org.jboss.ejb.client.RandomDeploymentNodeSelector@37374a5e
      deployment node selector selected qwaevisz-mac node for
      appname=diskstoreapp,modulename=dsservicemodule,distinctname=
2 2017-11-08 16:48:53 INFO  DiskClient:28 - DiskStub
      [reference=WAM124, author=Mozart, title=Requiem Mass in D
      minor, category=ROCK, price=2850.0, numberOfSongs=4]
3
4 DiskStub [reference=WAM124, author=Mozart, title=Requiem Mass in
      D minor, category=ROCK, price=2850.0, numberOfSongs=4]
5
6 2017-11-08 16:48:53 DEBUG RemoteNamingStoreV1:263 - Channel end
      notification received, closing channel Channel ID c1d5c67d
      (outbound) of Remoting connection 5ed31789 to
      localhost/127.0.0.1:4447
7 2017-11-08 16:48:53 INFO  remoting:445 - EJBCLIENT000016: Channel
      Channel ID d1b9a950 (outbound) of Remoting connection
      5ed31789 to localhost/127.0.0.1:4447 can no longer process
      messages
```

# Alternatív ORM megoldások

## Van-e élet a JPA-n kívül?

A *BookStore* project során megismerkedtünk a JPA-vel "Hello World" szinten. Ezt a Java EE standard részeként élő és fejlődő rendszert fogjuk a későbbiekben is használni, nagyvállalati környezetben ismerete elengedhetetlen. A JPA-nak azonban van egy viszonylag nagyobb tanulási görbéje, illetve - ami talán még veszélyesebb - egy olyan jellemzője, mi szerint komolyabb elméleti (itt értve ANSI SQL, ORM és JPA ismeretek hiányát) tudás nélkül is el lehet benne érni "futó kódrészleteket" (melyeket alkalmazásnak szándékosan nem neveznék). Főleg ez utóbbi miatt elég sok project belefut abba, hogy a JPA által generált lekérdezések, az *entity manager* által végrehajtott műveletek nem optimálisak, sőt a rendszer *bottleneck*-jeként hivatkoznak rá.

# Alternatív ORM megoldások

## type-safe, native SQL, JavaEE kompatibilitás

A *DiskStore* project bemutat egy alternatív ORM megoldást, mely **támogatja a Java EE integrációt** és **type-safe** kódolást biztosít a JDBC felett. Előnye hogy kizárólag ANSI SQL ismeret szükséges hozzá, sokkal gyorsabban és könnyebben átlátható, ezáltal optimalizálható, mellőzi a gazdag automatizmust, és az ebből eredő problémákat. Nagy hátránya hogy **native SQL** utasításokat használ (ezáltal *nem vendor független*) és egy professzionális JPA alapú rendszerrel összehasonlítva egy idő után borzasztó kényelmetlen és számos *boiler plate* kódot eredményez(het).

Bár ORM nélkül, egyszerűen a JDBC implementáció segítségével is tudunk adatbázis műveleteket végezni Java EE környezetben, ez lehetőleg sose legyen végcél (egy nem type-safe kódbázis karbantartása nagyvállalati környezetben szinte lehetetlen).

http://www.mybatis.org/mybatis-3/
Verzió: **3.4.6-SNAPSHOT** (2017-08-20)
Artifact URI: `org.mybatis:mybatis:3.4.6-SNAPSHOT`
A perzisztencia konfigurációs állománya egy `config.xml`, mely felelősségben hasonlatos a JPA-nál megismert `persistence.xml`-hez. Ezen xml dokumentum hivatkoz(hat) számos `mapper.xml`-re, melyben natív SQL utasításokat definiálhatunk. Az XML alapú konfigurációs elemek jelentős része itt is megtalálható annotáció formájában is.

Java EE integrációhoz szükséges egy **MyBatis CDI** függőség használata is
http://www.mybatis.org/cdi/
Verzió: **1.0.2** (2017-10-13)
Artifact URI: `org.mybatis:mybatis-cdi:1.0.2`
A CDI része a Java EE-nek, és később részletesebben lesz szó róla. Egyelőre arra figyeljünk majd, hogy a `beans.xml` állományunk elérhető legyen a *classpath*-on.

```
1 <transactionManager type="MANAGED">
2   <property name="closeConnection" value="true" />
3 </transactionManager>
```

TransactionManager type lehetséges értékei:

- **JDBC**: A JDBC commit/rollback lehetőségeit direktben használja
- **MANAGED**: A MyBatis semmit nem fog ez esetben a tranzakcióval kezdeni (nem lesz soha commit vagy rollback)

```
1 <dataSource type="POOLED">
2   <property name="driver" value="${driver}"/>
3   <property name="url" value="${url}"/>
4   <property name="username" value="${username}"/>
5   <property name="password" value="${password}"/>
6 </dataSource>
```

Datasource type lehetséges értékei:

- **UNPOOLED**: minden egyes alkalommal nyitja/csukja a kapcsolatot
- **POOLED**: webalkalmazás fejlesztésnél gyakran használt, hatékony
- **JNDI**: a datasource-ot egy JavaEE container biztosítja JNDI-on keresztül (ezt fogjuk alkalmazni)

ds-persistence | src | main | resources | mybatis-config.xml

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <typeAliases>
    <typeAlias type="hu.qwaevisz.diskstore.persistence.entity.Disk" alias="Disk" />
    <package name="hu.qwaevisz.diskstore.persistence.entity"/>
  </typeAliases>
  <typeHandlers>
    <typeHandler handler="org.apache.ibatis.type.EnumOrdinalTypeHandler"
        javaType="hu.qwaevisz.diskstore.persistence.entity.trunk.DiskCategory"/>
  </typeHandlers>
  <environments default="development">
    <environment id="development">
      <transactionManager type="MANAGED">
        <property name="closeConnection" value="true" />
      </transactionManager>
      <dataSource type="JNDI">
        <property name="data_source" value="java:jboss/datasources/diskstoreds" />
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="hu/qwaevisz/diskstore/persistence/mapper/DiskMapper.xml" />
  </mappers>
</configuration>
```

mybatis-config.xml

- **Type alias**: a `mapper.xml` állományokban a típusok *full qualified* neve helyett egy rövidebb változat is használható, ha megadunk egy `alias`-t. Annotáció is létezik minderre `@Alias` néven.
- **Type handler**: ha egy típus átalakítása nem egyértelmű (Java $\rightarrow$ SQL), akkor *type handler*-t szükséges készíteni. Ha egy *enum* értéknek az *ordinal*-ját szeretnénk táblába beszúrni a neve helyett, erre létezik egy előre készített *type handler* (`org.apache.ibatis.type.EnumOrdinalTypeHandler`), melyet a példában használunk is.
- **Environment**: A tranzakció kezelés és az adatforrás konfigurációja.
- **Mapper**: konfigurációs állományok listája, melyben a natív SQL-ek szerepelnek (illetve a *resultMap*-ek). Itt lehet megadni *mapper osztály*okat is, amennyiben a konfiguráció annotáció alapú.

# Disk entitás

```java
package hu.qwaevisz.diskstore.persistence.entity;
[..]
@Alias("Disk")
public class Disk {

  private Integer id;
  private String reference;
  private String author;
  private String title;
  private DiskCategory category;
  private Double price;
  private Integer numberOfSongs;

  public Disk() {
    [..]
  }

  [..]
}
```

Disk.java

## Mapper

A konfiguráció által meghivatkozott `mapper.xml` állományok natív SQL utasításokat, illetve *resultMap*-eket tartalmaznak a megadott névtér alatt. A **MyBatis 3** feladata alapvetően az, hogy az entitásokból és a `mapper.xml` állományokból az adatmanipulációhoz szükséges *boilerplate* kódokat elkészítse (attól lesz a történet *type-safe*, hogy ezen generált kódrészeket nem nekünk kell karbantartanunk).

`ds-persistence | src | main | resources | hu | qwaevisz | diskstore | persistence | mapper | DiskMapper.xml`

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
       "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
3  <mapper namespace="hu.qwaevisz.diskstore.persistence.mapper.DiskMapper">
4
5    <resultMap [..]>[..]</resultMap>
6    <select [..]>[..]</select>
7    <insert [..]>[..]</insert>
8    <update [..]>[..]</update>
9    <delete [..]>[..]</delete>
10
11 </mapper>
```

DiskMapper.xml

# Mapper állomány - count

## core input, core output

```
1 <select id="count" parameterType="String" resultType="int">
2   SELECT COUNT(1)
3   FROM disk
4   WHERE disk_reference = #{reference}
5 </select>
```

DiskMapper.xml

Ezzel a deklarációval egy `count` nevű metódust definiáltunk, melynek `String` bementi paramétere lesz és egy `int` értékkel fog visszatérni.

# Mapper állomány - readByReference

core input, entity output

```
1  <select id="readByReference" parameterType="String"
       resultType="Disk">
2    SELECT
3      disk_id AS id,
4      disk_reference AS reference,
5      disk_author AS author,
6      disk_title AS title,
7      disk_diskcategory_id AS category,
8      disk_price AS price,
9      disk_number_of_songs AS numberOfSongs
10   FROM disk
11   WHERE disk_reference = #{reference}
12 </select>
```

Ezzel a deklarációval egy `readByReference` nevű metódust definiáltunk, melynek `String` bementi paramétere lesz és egy `Disk` entitás példányával fog visszatérni. Mivel a `config.xml`-ben definiáltunk egy *alias*-t, elegendő csupán `Disk`-et írni az entitás *full qualified* neve helyett.
A lekérdezésben használt **AS** hozza létre a kapcsolatot az entitás *mutator* (setter) metódusával (Java bean elnevezési szabályok szerint).

# Mapper állomány - readAll

- input, list of resultMap output

```
1  <resultMap type="Disk" id="DiskResult">
2    <id property="id" column="disk_id" />
3    <result property="reference" column="disk_reference" />
4    <result property="author" column="disk_author" />
5    <result property="title" column="disk_title" />
6    <result property="category" column="disk_diskcategory_id" />
7    <result property="price" column="disk_price" />
8    <result property="numberOfSongs" column="disk_number_of_songs"
         />
9  </resultMap>
10
11 <select id="readAll" resultMap="DiskResult">
12   SELECT * FROM disk
13 </select>
```

Egyedi (pl. több táblát érintő, vagy aggregált, esetleg egyedi asszociációkat tartalmazó) lekérdezés esetén nem tudjuk direktben entitásként definiálni a visszatérési értéket. Ezt a problémát hidalja át a `resultMap` definiálásának lehetősége. Itt csupán egy "hello world" mintát mutat, lehetőségei ennél sokkal szélesebbek. Értelemszerűen a `property` attribútum a megadott `type` mögött egy *mutator* metódust vár el.

# Mapper állomány - insert

## entity input, - output

```
<insert id="create" parameterType="Disk" useGeneratedKeys="true" keyProperty="id">
  INSERT INTO disk (
    disk_reference,
    disk_author,
    disk_title,
    disk_diskcategory_id,
    disk_price,
    disk_number_of_songs
  ) VALUES (
    #{reference},
    #{author},
    #{title},
    #{category},
    #{price},
    #{numberOfSongs}
  )
</insert>
```

Ezen deklaráció által **create** néven fogunk tudni elérni egy `Disk` bementi értékkel bíró metódust. A bemenet különféle értékeinek itt már jelentősége van, pl. a `{#author}` egy `getAuthor()` *accessor* metódust vár el a bemeneti entitásban.
A példában az ID oszlop értékét `sequence` fogja generálni, a natív lekérdezésbe így nem szerepeltetjük az ID oszlopot. Ezt a `useGeneratedKeys` attribútummal tudjuk jelezni.

# Mapper állomány - update és delete

## hiányzó CRUD műveletek bemutatása

```
<update id="update" parameterType="Disk">
  UPDATE disk SET
    disk_reference = #{reference},
    disk_author = #{author},
    disk_title = #{title},
    disk_diskcategory_id = #{category},
    disk_price = #{price},
    disk_number_of_songs = #{numberOfSongs}
  WHERE disk_id = #{id}
</update>
```

```
<delete id="delete" parameterType="int">
  DELETE FROM disk WHERE disk_id = #{id}
</delete>
```

# Mapper interface

Az interface-ben megadott *abstract* metódusok pontosan illeszkednek a mögöttes `xml` konfigurációban definiált műveletekre. Ellenkező esetben futás idejű hibát kapunk (itt ez JDBC-s hatás, mely messze nem hibatűrő, azonban mégis kezelhetőbb, mintha minden Java-ban lenne).

```
1  package hu.qwaevisz.diskstore.persistence.mapper;
2
3  import java.util.List;
4  import hu.qwaevisz.diskstore.persistence.entity.Disk;
5
6  public interface DiskMapper {
7
8    int count(String reference);
9    int create(Disk disk);
10   Disk readById(Integer id);
11   Disk readByReference(String reference);
12   List<Disk> readAll();
13   int update(Disk disk);
14   int delete(Integer id);
15 }
```

DiskMapper.java

# MyBatis konfiguráció betöltése

A konfig állomány betöltésének egyszer meg kell történnie *runtime*. Ezt hivatott megoldani eme *Provider* osztály `@Produces` annotációval ellátott metódusa. A két annotáció a standard CDI része, nincs köze a *MyBatis 3* library-hoz. Céljuk hogy bárhol elérhető legyen az `SqlSessionFactory` inject-álása, és az első ilyen esetben ezen metódus fusson le. A CDI-ról később lesz szó.

```
1  package hu.qwaevisz.diskstore.persistence.config;
2  [..]
3  import javax.enterprise.context.ApplicationScoped;
4  import javax.enterprise.inject.Produces;
5  import org.apache.ibatis.io.Resources;
6  import org.apache.ibatis.session.SqlSessionFactory;
7  import org.apache.ibatis.session.SqlSessionFactoryBuilder;
8  [..]
9  public class SqlSessionFactoryProvider {
10
11   @Produces
12   @ApplicationScoped
13   public SqlSessionFactory produceFactory() throws IOException {
14     final InputStream inputStream = Resources.getResourceAsStream("mybatis-config.xml");
15     final SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(inputStream);
16     return factory;
17   }
18
19 }
```

`SqlSessionFactoryProvider.java`

# MyBatis JavaEE környezetben

```java
package hu.qwaevisz.diskstore.persistence.service;
[..]
import javax.inject.Inject;
import org.mybatis.cdi.Mapper;
[..]
@Stateless(mappedName = "ejb/diskService")
public class DiskServiceImpl implements DiskService {

  @Inject
  @Mapper
  private DiskMapper mapper;

  @Override
  public Disk readByReference(final String reference) throws
       PersistenceServiceException {
    LOGGER.debug("Read Disk by reference (" + reference + ")");
    try {
      return this.mapper.readByReference(reference);
    } catch (final Exception e) {
      LOGGER.error(e, e);
      throw new PersistenceServiceException("Failed to read Disk by reference (" +
           reference + ")! " + e.getMessage(), e);
    }
  }
  [..]
}
```

A `@Inject` a CDI megfelelője a `@EJB` annotációnak, míg a `@Mapper` itt egy CDI qualifernek használt annotáció, melyet a *MyBatis CDI* library definiál.

DiskServiceImpl.java