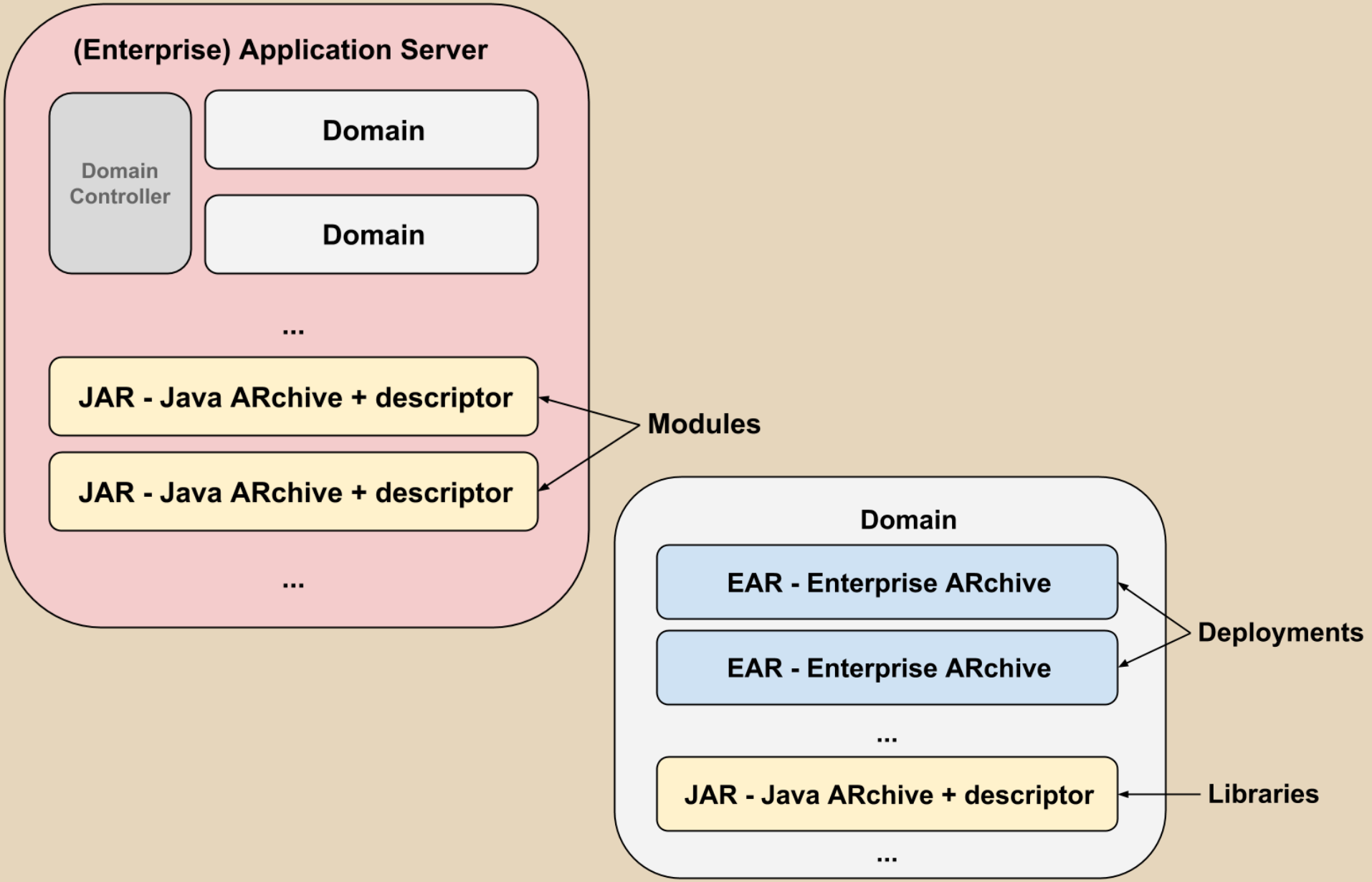
BookStore #gradle

Enterprise Application, Git, EJB, EAP/EAS, Logging, PostgreSQL/MySQL, JPA, Integration testing

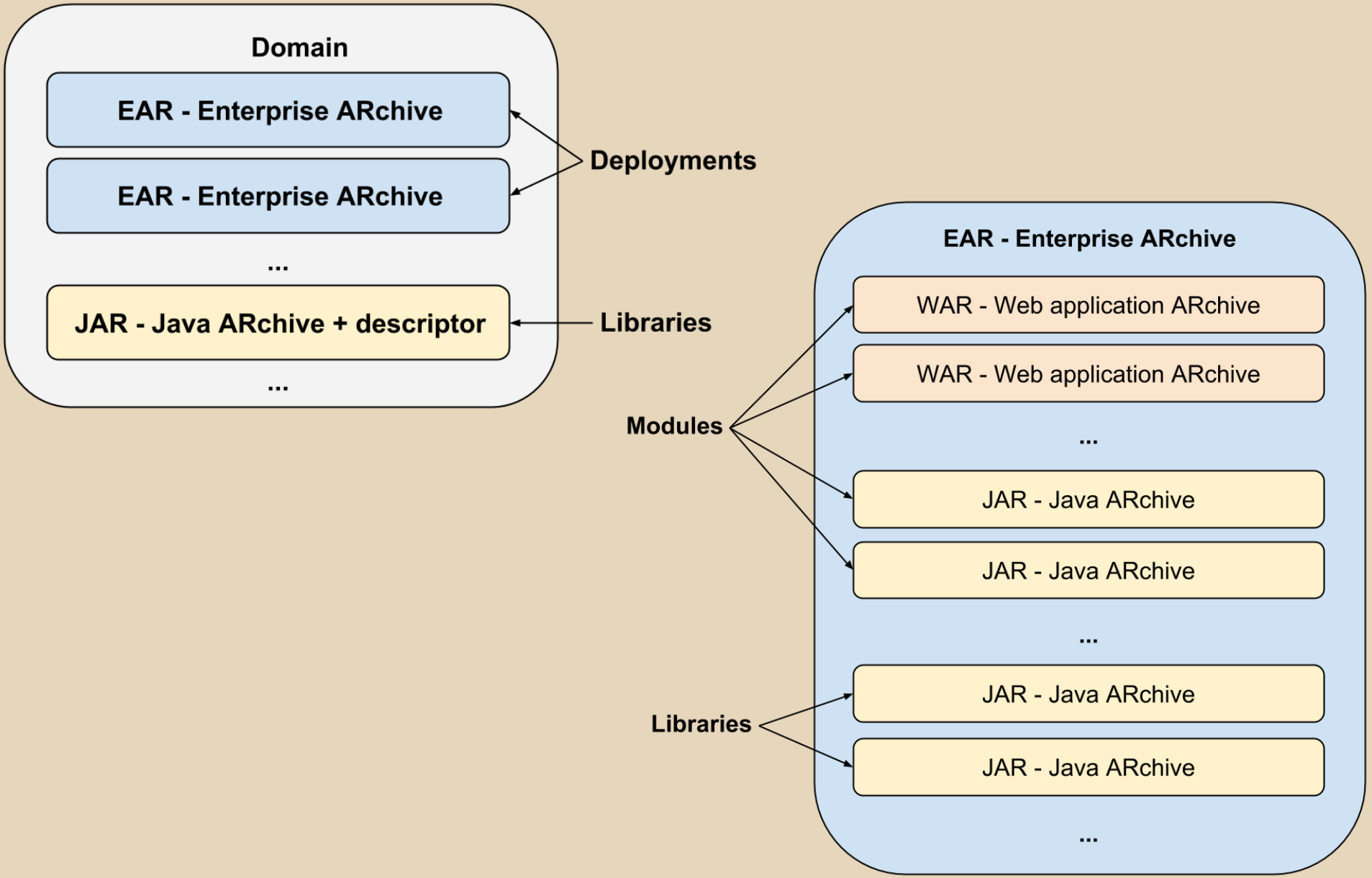
Óbudai Egyetem, Java Enterprise Edition
Műszaki Informatika szak
Labor 3

Bedők Dávid
2018-01-17
v1.1

EAS - Domain struktúra



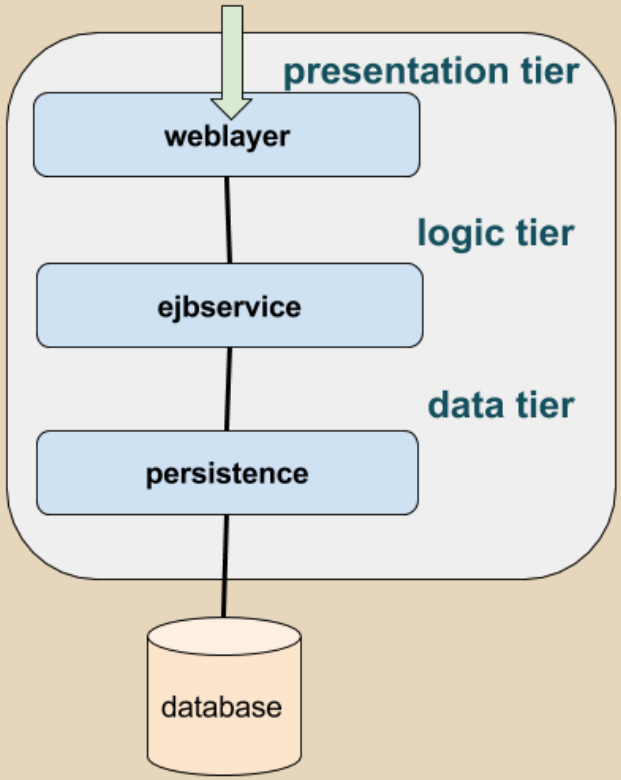
EAS - Deployment struktúra



Feladat: hozzunk létre egy alkalmazást, mely egy könyvesbolt könyveinek lekérdezésére alkalmas.
A feladatot két részletben valósítjuk meg. Első körben a persistence réteget mock-olni/fake-elni fogjuk.

Alkalmazandó technikák:

- Java Enterprise Edition 6
  - EJB API 3.1
  - Servlet API 3.0.1
  - JPA 2.0
- JBoss 6.4 / WebLogic 12.1.3



In medias res

Teszteljük le a fejlesztési környezetünket

```
> cd [GIT]\gradle\jboss\booklight\
> gradle clean build
..
BUILD SUCCESSFUL
```

Új EAR csomag:
[GIT]\gradle\jboss\booklight\build\libs\booklight.ear

```
> cd [JBOSS-HOME]\bin
> standalone.[bat|sh]
```

Másoljuk be az elkészült **booklight.ear** állományt a [JBOSS-HOME]\standalone\deployments könyvtárba.
http://localhost:8080/bl-weblayer/BookPing

```
BookStub [isbn=978-0441172719, author=Frank Herbert, title=Dune,
category=SCIFI, price=3500.0, numberOfPages=896]
```

- Projekt hierarchia létrehozása
- **Enterprise JavaBean** (`ejb`) alapok mint **business layer**
- Egyszerű servlet mint **weblayer**
- Mockolt adatok mint **persistence layer**

Enterprise Application Servers (EAS)

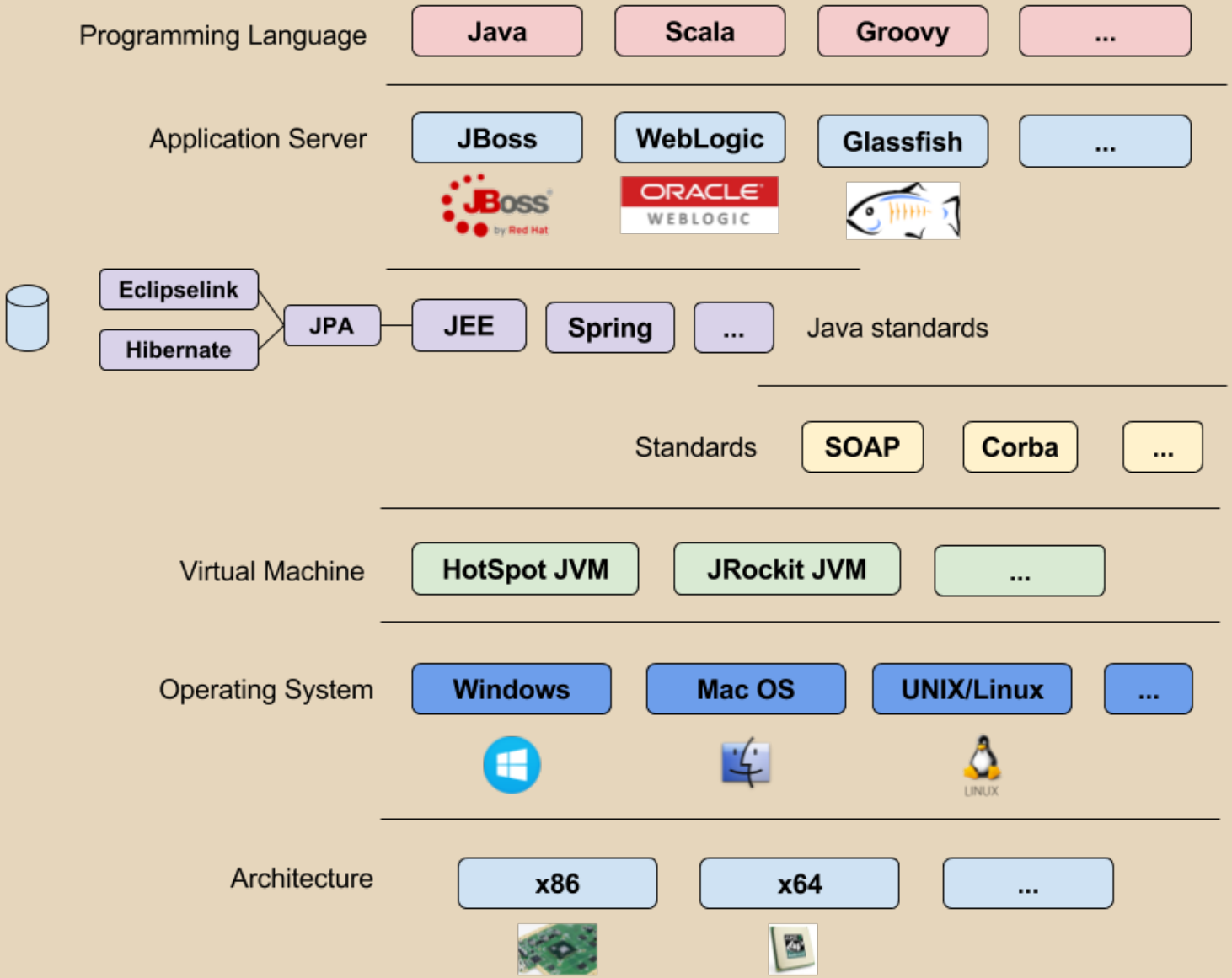
- RedHat JBoss 6.4
- Oracle WebLogic 12.1.3

Másik EAS választásának lehetősége

A JavaEE-ben megvalósított alkalmazások minden olyan Java alkalmazás szerveren futnak, melyek az adott verziójú JavaEE-t támogatják. Ez az átjárhatóság tovább erősíti a Java egyébként is **cross-platform** tulajdonságát (bár attól hogy egy nyelv cross-platform, még nem lenne természetes ekkora komplexitású alkalmazások mozgatása AS-ek között).

A feladatot két különböző EAS-ra fogjuk elkészíteni/deploy-olni.

Cross-Platform Java



Repository

`[gradle|maven]\jboss\booklight`

```
1 > gradle clean build
```

Megjegyzés: a gradle `build`-je `ear plugin` esetén **ear**-t, `war plugin` esetén **war**-t, `java plugin` esetén **compile**-t fog végrehajtani (ezek mind függnek a `build`-től).

A blueprint repository frissítése (Git `pull`) szükséges, ha az `origin/master` (branch) változott. E művelet előtt a lokális változásainkat el kell dobni.

Working copy kiválasztása (root könyvtárban)

```
1 > git checkout .
```

Lokális módosítások eldobása

```
1 > git reset --hard
```

Új állományok/könyvtárak eldobása

```
1 > git clean -fd
```

Változások leszedése a **master**-ről

```
1 > git pull
```

http://gitextensions.github.io/
https://github.com/gitextensions

Verzió: **2.50.02**
OS: **Windows** (NIX esetén Mono 5.0 szükséges)

Grafikus felületet biztosít a `Git` parancsok végrehajtásához. Nem csak eszköz, tanítja is a kiadott parancsokat.
A Git használat nem lineáris, és ugyanaz a feladat többféleképpen is megoldható. Minden külső eszköz használata e végett magában foglal némi kockázatot (nem biztos hogy a tool azt hajtja végre, amit mi szeretnénk).
Alapfunkciók esetén szinte kockázatmentes.

https://www.sourcetreeapp.com/

Verzió: **2.6.3a**
OS: **Windows / Mac OS**

Elegáns, modern felület, néhol kicsit körülményes (pl. git log/blame esetén), felületét szokni kell, de azt követően hatékony eszköz. Windows rendszereken korábban lassú volt. Nem tanítja a `git` parancsokat, mint pl. a GitExtensions.

Git Commit és Push

Új fejlesztések/hibajavítások megosztása

Állapot lekérdezése:

```
> git status
```

Stage

```
> git add [FILENAME]
```

Commit (-a/–all, -m/–message)

```
> git commit -a -m "[COMMIT_MESSAGE]"
```

Változások felküldése a master-re

```
> git push
```

Több project, hierarchiába szervezve

- WebLayer
  - gradle **java** és **war** plugin → `war`
  - EAR **web module**
- EJBService
  - gradle **java** → `jar`
  - EAR **ejb module**
- Root project
  - gradle **ear** plugin → `ear`

WAR - Web application ARchive

Könyvtár struktúra

```
META-INF/
    MANIFEST.MF
WEB-INF/
    classes/
        hu/
            qwaevisz/
                demo/
                    Lorem.class
                    Ipsum.class
    lib/
        dummy.jar
    web.xml
index.html
logo.jpg
base.css
```

public resources (alkönyvtárakba elhelyezhetőek)

```
1 <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
2         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
              http://java.sun.c
4 [...]
5 </web-app>
```

web application deployment descriptor

web.xml

Könyvtár struktúra

```
META-INF/
    MANIFEST.MF
    application.xml
lib/
    dummy.jar
xyz.jar
abc.war
```

3rd party library

```
1  <?xml version="1.0"?>
2  <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">
3    <display-name>sample</display-name>
4    <module>
5      <ejb>xyz.jar</ejb>
6    </module>
7    <module>
8      <web>
9        <web-uri>abc.war</web-uri>
10       <context-root>abcwebapp</context-root>
11     </web>
12   </module>
13   <library-directory>lib</library-directory>
14 </application>
```

application.xml

Könyvtár struktúra

```
booklight/
    bl-ejbservice/
        src/
            main/
                java/
                resources/
                    META-INF/
                        ejb-jar.xml
        build.gradle
    bl-weblayer/
        src/
            main/
                java/
                webapp/
                    WEB-INF/
                        web.xml
        build.gradle
build.gradle
settings.gradle
```

EJB module - Deployment Descriptor

ejb-jar.xml

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

```
1 <ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
3   version="3.0">
4
5 </ejb-jar>
```

ejb-jar.xml

Kötelező elem?

Bár a Gradle és a JBoss nem követeli meg az üres `ejb-jar.xml` jelenlétét a classpath-on, érdemes szerepeltetni az átjárhatóság végett (pl. Maven megköveteli a jelenlétét). A *deployment descriptor*-ban definiálható szabályok jelentős részét ma már annotációk segítségével a forráskódban helyezzük el. A leíró xml precedenciája mindig magasabb (felül lehet írni az annotációk által szabályozott elemeket).

Root EAR project

Gradle EAR plugin

Az ear plugin dependency configuration lehetőségei: **deploy** és **earlib**.

```
1 apply plugin: 'eclipse'
2 apply plugin: 'ear'
3
4 dependencies {
5   deploy project('bl-ejbservice')
6   deploy project(path: 'bl-weblayer', configuration: 'archives')
7 }
```

Project project(String path)
Project project(String path, String configuration)

build.gradle

```
1 include 'bl-ejbservice'
2 include 'bl-weblayer'
```

settings.gradle

EJB Service subproject

Gradle Java plugin

```
1  apply plugin: 'eclipse'
2  apply plugin: 'java'
3
4  repositories { mavenCentral() }
5
6  jar { archiveName 'bl-ejbservice.jar' }
7
8  dependencies {
9    compile group: 'javax', name:'javaee-api', version:'6.0'
10 }
```

Pontosan ez az alapértelmezett érték, így elhagyható a teljes blokk.

build.gradle

WebLayer subproject

A **war** plugin *dependency configuration* lehetőségei:

- **providedCompile**: kell a fordításhoz, de a container biztosítja
- **providedRuntime**: a container biztosítja

Használható a **java** plugin **compile** dc-je is: kell a fordításhoz, de a container nem biztosítja (az *war* artifact `WEB-INF\lib` könyvtárába fog kerülni).

```
1  apply plugin: 'eclipse'
2  apply plugin: 'java'
3  apply plugin: 'war'
4
5  repositories { mavenCentral() }
6
7  war { archiveName 'bl-weblayer.war' }
8
9  dependencies {
10   providedCompile project(':bl-ejbservice')
11   providedCompile group: 'javax', name:'javaee-api', version:'6.0'
12 }
```

`build.gradle`

A ":" jelentése egy project neve előtt azt jelzi, hogy a projectet a root project-től kiindulva kell keresni (*absolute* elérés lesz, nem pedig *relative*).

Enterprise JavaBeans

- Szerveroldali üzleti komponensek (back-end services)
- JSR19, JSR152, JSR220, JSR318, JSR345
- IBM (1997), majd Sun Microsystems (1999)
- Egyfajta "best-practice" annak érdekében, hogy az üzleti értéket "kelljen csak" lefejleszteni, ne a "tipikus" keret dolgokat (boilerplate elemeket).
  - Tranzakciókezelés
  - Konkurrenciakezelés
  - Aszinkron metódushívás
  - Eseménykezelés
  - Java Message Service integráció (Message Driven Beans)
  - Perzisztencia integráció támogatása (de már nem része a perstence az EJB specifikációnak)
- Verziók
  - EJB1.x: minden "remote"
  - EJB2.x: nagyon "kényelmetlen", túlbonyolított
  - EJB3.x: a Spring Framework és a Hibernate tapasztalatai alapján újraalkotott elképzelés (POJO-kal dolgozik mint a Spring FW és szakít az entity bean-ekkel (inkább támogatja a perzisztens réteget, nem megvalósítja))

EJB típusok

- **Session Beans**
  - **Stateless SB** (SLSB)
    - Állapotmentes
  - **Stateful SB** (SFSB)
    - Állapottartó
    - Aktiválás/Passziválás
    - Fontos a szerializálhatóság
  - **Singleton SB** (SSB)
    - "Egyke"
- **Message Driven Beans** (MDB)
  - Üzenetvezérelt
  - Elsősorban aszinkron feldolgozás
  - Nincs kliens interface
- **Entity Beans**
  - Deprecated, EJB 3.x-től a JPA helyettesíti

Business Layer

Definiáljunk néhány üzleti metódust:

- ISBN[1] alapján egy könyv adatainak biztosítása
  - `BookStub getBook ( String isbn )`
- Keresési feltételek alapján a feltételnek megfelelő könyvek adatainak kinyerése
  - `List<BookStub> getBooks ( BookCriteria criteria )`
- `FacadeException`: ha az üzleti kérés nem teljesíthető

Definiáljunk az ezekhez szükséges DTO[2]-kat:

- BookStub: isbn (`String`), author (`String`), title (`String`), category (`BookCategoryStub`), price (`double`), numberOfPages (`int`)
- BookCategoryStub: `SCIFI`, `LITERATURE`, `HISTORICAL`, `PHILOSOPHY`
- BookCriteria: author (`String`), title (`String`), minPrice (`int`), maxPrice (`int`)

[1] International Standard Book Number
[2] Data Transfer Object

Domain osztályok - Könyv kategóriák

```java
package hu.qwaevisz.booklight.ejbservice.domain;
public enum BookCategoryStub {

  SCIFI("Sci-Fi"),
  LITERATURE("Literature"),
  HISTORICAL("Historical"),
  PHILOSOPHY("Philosophy");

  private final String label;

  private BookCategoryStub(String label) {
    this.label = label;
  }

  public String getLabel() {
    return this.label;
  }
  public String getName() {
    return this.name();
  }
}
```

BookCategoryStub.java

Domain osztályok - Könyv

```
1  package hu.qwaevisz.booklight.ejbservice.domain;
2  public class BookStub {
3    private String isbn;
4    private String author;
5    private String title;
6    private BookCategoryStub category;
7    private double price;
8    private int numberOfPages;
9
10   public BookStub(String isbn, String author, String title,
         BookCategoryStub category, double price, int numberOfPages)
         {
11     this.isbn = isbn;
12     this.author = author;
13     this.title = title;
14     this.category = category;
15     this.price = price;
16     this.numberOfPages = numberOfPages;
17   }
18   [...]
19 }
```

BookStub.java

Domain osztályok - Keresési feltételek

```java
package hu.qwaevisz.booklight.ejbservice.domain;

public class BookCriteria {

  private String author;
  private String title;
  private BookCategoryStub category;
  private int minimumPrice;
  private int maximumPrice;

  public BookCriteria() {
  }

  [...]
}
```

BookCriteria.java

Domain osztályok - Dobható üzleti kivétel

```java
package hu.qwaevisz.booklight.ejbservice.exception;

public class FacadeException extends Exception {

  private static final long serialVersionUID = 1L;

  public FacadeException(String message) {
    super(message);
  }

  public FacadeException(String message, Throwable cause) {
    super(message, cause);
  }

}
```

FacadeException.java

Stateless Session Bean

Local interface

```
1  package hu.qwaevisz.booklight.ejbservice.facade;
2
3  import java.util.List;
4  import javax.ejb.Local;
5  import hu.qwaevisz.booklight.ejbservice.domain.BookCriteria;
6  import hu.qwaevisz.booklight.ejbservice.domain.BookStub;
7  import hu.qwaevisz.booklight.ejbservice.exception.FacadeException;
8
9  @Local
10 public interface BookFacade {
11
12   BookStub getBook(String isbn) throws FacadeException;
13
14   List<BookStub> getBooks(BookCriteria criteria) throws FacadeException;
15 }
```

BookFacade.java

`@Remote`: költséges és lassú (hálózati hívás, serializáció), RMI, eredeti koncepció csak ezt támogatta, különböző JVM-ek között használjuk
`@Local`: olcsó és gyors, azonos JVM-en belül (elsősorban adott EAR-on belül)

Stateless Session Bean

Implementation

```java
package hu.qwaevisz.booklight.ejbservice.facade;

[...]

@Stateless(mappedName = "ejb/bookFacade")
public class BookFacadeImpl implements BookFacade {

  @Override
  public BookStub getBook(String isbn) throws FacadeException {
    return new BookStub(isbn, "Frank Herbert", "Dune", BookCategoryStub.SCIFI, 3500,
        896);
  }

  @Override
  public List<BookStub> getBooks(BookCriteria criteria) throws FacadeException {
    List<BookStub> stubs = new ArrayList<BookStub>();
    stubs.add(new BookStub("978-0441172719", "Frank Herbert", "Dune",
        BookCategoryStub.SCIFI, 3500, 896));
    stubs.add(new BookStub("978-0684824710", "Daniel C. Dennett", "Darwin's dangerous
        idea", BookCategoryStub.PHILOSOPHY, 4500, 586));
    return stubs;
  }

}
```

Mock implementáció (literálokat ad vissza)

BookFacadeImpl.java

`@Stateless`: Stateless Session Bean EJB típus definiálása

- `name`: ejb neve
  - JNDI lookup során használatos
  - alapértelmezetten az *unqualified name* (pl.: `BookFacadeImpl`)
- `mappedName`: global JNDI név
  - gyártó specifikus (vendor specific)
- `description`: leírás

Servlet

Egyelőre csak tesztelési célból

```java
package hu.qwaevisz.booklight.weblayer.servlet;
[...]
@WebServlet("/BookPing")
public class BookPingServlet extends HttpServlet {

  @EJB
  private BookFacade facade;

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
       ServletException, IOException {
    response.setCharacterEncoding("UTF-8");
    final PrintWriter out = response.getWriter();
    try {
      final BookStub book = this.facade.getBook("978-0441172719");
      out.println(book.toString());
    } catch (final FacadeException e) {
      out.println(e.getLocalizedMessage());
    }
    out.close();
  }
}
```

BookPingServlet.java

EJB proxy inject-álása

@EJB annotáció

`@EJB`: Session Bean proxy beszúrása

- Csak az EJB "érdekszférájában" használható
  - Minden EJB ide tartozik
  - Minden Servlet ide tartozik (`@WebServlet`)
  - stb..
- `beanName` vagy `lookup`: az EJB feloldásához használható attribútumok
  - A kettő közül egy időben csak az egyik használható
  - Ha csak egy implementáció van, akkor alkalmazáson belül egyik attribútumot sem kell használni (típus alapján megtalálja a container a megfelelőt).
  - `beanName`: A Session bean `name` értékére refereál
  - `lookup`: A Session bean JNDI nevére refereál

Az Óbudai Egyetem "Modern programozási nyelvek" laborjának ide vonatkozó részei szükséges és elégséges feltétele a jelen labor során bemutatott frontend fejlesztésnek.

A tárgy nem hivatalos oldala:
http://users.nik.uni-obuda.hu/bedok.david/jse.html

A tárgy hivatalos oldala:
http://users.nik.uni-obuda.hu/java/

http://www.jboss.org/

Install: unzip

Enterprise **J**ava**B**eans **O**pen **S**ource **S**oftware → EJBoss

→ JBoss (az EJB® Sun trademark volt)

JEE 7 compliant verzió: **v7.0.0** (2016.05.10.)

JEE 6 compliant verzió: **v6.4.0** (2015.04.15.)

Kereskedelmi termék!

Standalone / Domain mode

Tartalma: https://access.redhat.com/articles/112673

- RESTEasy - 2.3.10.Final
- Hibernate Core - 4.2.18.Final
- Hibernate JPA 2.0 API - 1.0.1.Final
- JSF2 - 2.1.28.Final
- HornetQ - 2.3.25.Final (EAP 7.x → JBoss A-MQ)
- JBoss Logging - 3.1.4.GA
- JAXB - 2.2.5-redhat-9
- Apache Web Server - 2.2.26
- ...

http://wildfly.org/
Install: unzip
JEE 7 compliant verzió: **v11.0.0.Final** (2017.10.23.)
JEE 7 compliant verzió: **v10.1.0.Final** (2016.08.19.)
JEE 7 compliant verzió: **v8.0.0.Final** (2014.02.11.)
A JBoss **ingyenes** változata.
Standalone / Domain mode

JBoss vs. WildFly

A két alkalmazásszerver gyakorlati használata közel megegyezik, ha valaki az egyikkel dolgozott már, a másik vonatkozásában nem fog váratlan nehézségekbe belefutni.

Standalone mód ([JBOSS-HOME]\standalone):

- \configuration\standalone.xml
- \deployments
- logs\server.log

Standalone mód elindítása (standalone.xml):

```
1 > cd [JBOSS-HOME]\bin
2 > standalone.[bat|sh]
```

Standalone mód elindítása (custom.xml használata):

```
1 > [JBOSS-HOME]\bin\standalone.[bat|sh] custom.xml
```

Standalone server instance leállítása:

```
1 > [JBOSS-HOME]/bin/jboss-cli.[bat|sh] --connect command=:shutdown
```

http://www.oracle.com/technetwork/middleware/weblogic/

JEE 7 compliant verzió: **v12.2.1.3**
JEE 6 compliant verzió: **v12.1.3**
Kereskedelmi termék!
Az Oracle felvásárlását megelőzően a **BEA** terméke volt.

A JavaEE mellett a Spring Framework-öt is támogatja (a SF számára egy *Apache Tomcat*® is legtöbbször elégséges).

- Unzip (`wls1213_dev_update3.zip`)
  - pl.: `c:\work\wls12130\`
- Környezeti változó:
  - **MW_HOME** → `c:\work\wls12130\`
- Telepítő futtatása (*Run as Administrator*)
  - `[MW_HOME]\configure.cmd`
  - Do you want to configure a new Domain? Yes (Y)
  - username: **weblogic**
  - password: **AlmafA1#**
- WebLogic Server Administration Console
  - `http://localhost:7001/console`

Domain elindítása:

```
1 > [MW_HOME]\user_projects\domains\mydomain\startWebLogic.cmd
```

Domain leállítása:

- WebLogic Server Administration Console-ról illetve a console ablak terminálása (Ctrl+C)
  - Environment | Servers | Control tab
    - select “myserver” → Shutdown
      - When work completes vagy
      - Force Shutdown now

- WebLogic Server Administration Console
  - Deployments | Install..
    - Browse EAR
    - Ezt követően lehet ugyanezen path-ról egy-két kattintással frissíteni az alkalmazást (redeploy)

A WebLogic gazdag statisztikákat kezel, monitorozza a futó komponenseket. Érdemes az Admin Console-on megvizsgálni a lehetőségeket.

Autodeploy könyvtár:
`[MW_HOME]\user_projects\domains\mydomain\autodeploy\`

Megjegyzés: az auto deploy-olt alkalmazások nem törölhetőek/frissíthetőek A `Deployments` alól (a autodeploy könyvtárból kell kezelnünk ezt is).

[gradle|maven]\jboss\booklight\build\libs\booklight.ear

Másolás ide: [JBOSS-HOME]\standalone\deployments\

```
09:14:13,505 INFO  [org.jboss.as.server.deployment] (MSC service thread 1-6)
JBAS015876: Starting deployment of "booklight.ear" (runtime-name: "booklight.ear")
09:14:13,533 INFO  [org.jboss.as.server.deployment] (MSC service thread 1-6)
JBAS015973: Starting subdeployment (runtime-name: "bl-ejbservice.jar")
09:14:13,534 INFO  [org.jboss.as.server.deployment] (MSC service thread 1-5)
JBAS015973: Starting subdeployment (runtime-name: "bl-weblayer.war")
09:14:13,577 INFO
[org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeploymentUnitProcessor] (MSC
service thread 1-1) JNDI bindings for session bean named BookFacadeImpl in deployment
unit subdeployment "bl-ejbservice.jar" of deployment "booklight.ear" are as follows:
  java:global/booklight/bl-ejbservice/BookFacadeImpl!hu.qwaevisz.booklight.ejbservice.facad
  java:app/bl-ejbservice/BookFacadeImpl!hu.qwaevisz.booklight.ejbservice.facade.BookFacade
  java:module/BookFacadeImpl!hu.qwaevisz.booklight.ejbservice.facade.BookFacade
  java:global/booklight/bl-ejbservice/BookFacadeImpl
  java:app/bl-ejbservice/BookFacadeImpl
  java:module/BookFacadeImpl
09:14:13,646 INFO  [org.jboss.web] (ServerService Thread Pool -- 11) JBAS018210:
Register web context: /bl-weblayer
09:14:13,739 INFO  [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS015859:
Deployed "booklight.ear" (runtime-name : "booklight.ear")
```

server.log

http://localhost:8080/bl-weblayer/BookPing

```
BookStub [isbn=978-0441172719, author=Frank Herbert, title=Dune,
category=SCIFI, price=3500.0, numberOfPages=896]
```

A **BookStore** project teljes egészében tartalmazza a **BookLight** funkcionalitását, azonban az egyértelműség végett a projekt és a csomagok neve különböző.

```
[gradle|maven]\jboss\bookstore
```

Innentől kezdve a **BookStore** gyökere elkészíthető a **BookLight** alapján, és a fejlesztés a **BookStore** projektben folytatható.
A **BookStore** már tartalmazni fogja a *data tier* réteget (persistence layer).

Root project - Refactor

Redundancia csökkentése

```
1  [..]
2  ext {
3    testngVersion = '6.9.+'
4    jeeVersion = '6.0'
5    servletapiVersion = '3.0.1'
6  }
7
8  subprojects {
9    apply plugin: 'eclipse'
10   apply plugin: 'java'
11
12   repositories { mavenCentral() }
13
14   dependencies {
15     compile group: 'javax', name: 'javaee-api', version:
           jeeVersion
16     testCompile group: 'org.testng', name: 'testng', version:
           testngVersion
17   }
18 }
19 [..]
```

A `subprojects {...}` blokk minden gyermek build script-re érvényes (merge), és létezik egy `allprojects {...}` blokk is, mely mind az aktuális (ez esetben a *root*), mind a gyermek scriptekre érvényes.

Subprojects - Refactor

Egyedi eltérések meghatározása

bs-ejbservice subproject:

```
1 jar { archiveName 'bs-ejbservice.jar' }
```

build.gradle

Mivel ez az alapértelmezés, akár üres is lehetne ez a *build* file.

bs-weblayer subproject:

```
1 apply plugin: 'war'
2
3 war { archiveName 'bs-weblayer.war' }
4
5 dependencies {
6   providedCompile project(':bs-ejbservice')
7   providedCompile group: 'javax', name: 'javaee-api', version:
        jeeVersion
8 }
```

build.gradle

http://logging.apache.org/log4j/2.x/
http://logging.apache.org/log4j/1.2/

Log4j2 verzió: **v2.9.1**
Log4j verzió: **v1.2.17**

JBoss 6.4 egyelőre a Log4j 1.2.x-et támogatja, a WebLogic alapértelmezetten nem támogatja.

Standard?

A JavaEE-nek nem része a Log4J, a standard a **JDK Logging**-ot tartalmazza (ezt a JBoss és a WebLogic is támogatja természetesen).

Simple Logging Facade for Java (**SLF4J**): képes elfedni a különböző logging megoldásokat, és mindezeket dinamikusan kezelni (java.util.logging, logback, log4j).

- **Loggers**
  - Összerendeli a Java csomagot, a log level-t és az appendereket (1 logger N appender)
  - Egy ún. `Root Logger` mindig létezik.
- **Appenders**
  - Definiálja a log üzenet "pattern"-jét és a logolás típusát (pl. file, rolling file, e-mail, jms message, stb.).

Konfiguráció

Standalone alkalmazás esetében **log4j(2).xml** vagy **log4j(2).properties** állományban konfiguráljuk. A Jboss esetén a **Logging subsystem** csomagolja, és a **standalone.xml**-en keresztül konfiguráljuk.

- **TRACE**
  - Nagyon részletes adatok, csak kivételes esetben érdemes bekapcsolni (pl. ciklusmagok belsejének logjai).
- **DEBUG**
  - Hibakeresési/fejlesztési szint. Végigkövetik a log üzenetek az üzleti folyamatokat komponens szinten. A Debug logok segítségével reprodukálható pl. a felhasználó tevékenysége.
- **INFO**
  - Azok az üzenetek, melyek minden esetben jelenjenek meg a logban. Különösen fontos és semmiképpen sem gyakori események írhatóak ki ezen a szinten. Az info szintű üzenetek kb. a "hahó, itt vagyok" típusúak, belőlük pl. hibát keresni, reprodukálni nem lehet.
- **WARN**
  - Figyelmeztetés. Általában olyan esemény, mely azért nem hiba, mert az alkalmazás lekezeli valamilyen módon, vagy pl. a tranzakció végrehajtódik, de bizonyos korlátozásokkal.
- **ERROR**
  - Hiba. Olyan részletes leírást tartalmazzon, amennyire lehetséges. Ez lesz az az információ, amit a hibajegy javításakor birtokolni fogunk. Ez alapján kérni kell tudni a reprodukciót (mely során már kérhetünk debug logokat is).
- **FATAL**
  - Ritkán alkalmazzuk. Végzetes hiba. Az ügyfelek már az Error esetén is üvöltenek, nem érdemes fatális hibákkal ijesztgetni őket.

```java
public class BookPingServlet extends HttpServlet {

  private static final Logger LOGGER =
      Logger.getLogger(BookPingServlet.class);

  @Override
  protected void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException,
      IOException {
    LOGGER.info("Get Book by user");
    [..]
  }
}
```

BookPingServlet.java


```
1  public class BookFacadeImpl implements BookFacade {
2
3    private static final Logger LOGGER =
         Logger.getLogger(BookFacadeImpl.class);
4
5    @Override
6    public BookStub getBook(String isbn) {
7      if (LOGGER.isDebugEnabled()) {
8        LOGGER.debug("Get Book (isbn: " + isbn + ")");
9      }
10     [..]
11   }
12 }
```

alternatív megoldás (kevesebb kódsor):

```
LOGGER.debug(String.format("Get Book (isbn: %s)", isbn);
```

BookFacadeImpl.java

Fontos! Production kódban a DEBUG üzenetek se a teljesítményt, se a memóriát ne terheljék, mivel valós futás közben általában a DEBUG szintet nem naplózzuk. A bemutatott megoldás egy logikai ellenőrzés költségű, míg az alternatív megoldás egy String literált elhelyez a permspace-en. A String összefűzés futását mindenképpen kerüljük el ilyen esetekben (ebből StringBuilder-t készít a fordító, de feleslegesen optimalizálja a sort ha nem naplózzuk).

```
1  <server xmlns="urn:jboss:domain:1.7">
2    <extensions>
3      <extension module="org.jboss.as.logging"/>
4      [..]
5    </extensions>
6    <management>[..]</management>
7    <profile>
8      <subsystem xmlns="urn:jboss:domain:logging:1.5">
9        [APPENDERS DETAILS]
10       [LOGGER DETAILS]
11     </subsystem>
12     [..]
13   </profile>
14   <interfaces>[..]</interfaces>
15   <socket-binding-group>[..]</socket-binding-group>
16 </server>
```

standalone.xml

```
1  <console-handler name="CONSOLE">[..]</console-handler>
2  <periodic-rotating-file-handler
       name="FILE">[..]</periodic-rotating-file-handler>
3
4  <logger category="hu.qwaevisz">
5    <level name="DEBUG"/>
6  </logger>
7
8  <root-logger>
9    <level name="INFO"/>
10   <handlers>
11     <handler name="CONSOLE"/>
12     <handler name="FILE"/>
13   </handlers>
14 </root-logger>
15 <formatter name="PATTERN">[..]</formatter>
```

standalone.xml

A *Log4J* beállításához külön lépések szükségesek (alapértelmezetten nem támogatja a WebLogic, bár szállít hozzá wrapper-t).
Helyette **JDK Logging**-ot használjunk (JavaEE része, java.util.logging.*).

```
1 import java.util.logging.Level;
2 import java.util.logging.Logger;
3 [..]
4 private static final Logger LOGGER =
      Logger.getLogger(BookListController.class.getName());
5 [..]
6 LOGGER.info("Get All Books");
```

BookListController.java

JDK Logging vs Log4J

JDK Logging Level	Log4J Level	JDK Logging sample
FINEST	-	LOGGER.finest();
FINER	TRACE	LOGGER.finer();
FINE	DEBUG	LOGGER.fine();
CONFIG	-	LOGGER.config();
INFO	INFO	LOGGER.info();
WARNING	WARN	LOGGER.warning();
SEVERE	ERROR	LOGGER.severe();
-	ERROR	-

Ha a hiba mellé a kivételt is szeretnénk továbbadni (ez Log4J esetén egyszerű overload metódus):

```
1 LOGGER.log(Level.SEVERE, e.getMessage(), e);
```

```
1  [..]
2  ext {
3    [..]
4    log4jVersion = '1.2.17'
5  }
6
7  subprojects {
8    [..]
9
10   dependencies {
11     compile group: 'log4j', name: 'log4j', version: log4jVersion
12     [..]
13   }
14
15 }
16 [..]
```

build.gradle

PostgreSQL

Professzionális nyílt forráskódú adatbáziskezelő

http://www.postgresql.org/

Verzió: **v10.0** (2017.10.05.)

Verzió: **v9.6.5** (2017.09.31.)

Telepítés: installer

Alapértelmezett adatok: **5432** (port) és **postgres** (database)

Alapértelmezett felhasználó: **postgres** (user) és **root** (password)

Szerkesztő: **pgAdmin III** vagy **pgAdmin IV**

Környezeti változók:

- **PSQL_HOME** →`c:\ProgramFiles\PostgreSQL\9.4`
- **Path** módosítása → `%Path%;%PSQL_HOME%\bin`

Professzionális eszköz

A mellett, hogy a PostgreSQL hasonlóan professzionális management lehetőséggel rendelkezik, mint pl. az Oracle DB vagy a SAP Adaptive Server Enterprise (korábban Sybase ASE), létezik production környezetben rendszer melyben 4 TB adatot kezel.

Verzió lekérdezése

```
1 > psql --version
2 psql (PostgreSQL) 9.4.1
```

Csatlakozás (CLI)

```
1 > psql -d postgres -h localhost -p 5432 -U postgres
```

Csatlakozás és utasítások végrehajtása állományból

```
1 > psql -d postgres -h localhost -p 5432 -U postgres -f [FILENAME]
```

Oracle MySQL

A legnépszerűbb nyílt forráskódú adatbáziskezelő

https://www.mysql.com/

Letöltés: https://dev.mysql.com/downloads/

A Community verzió ingyenes (GPL).

Verzió: **v5.7.20**

Alapértelmezett adatok: **3306** (port)

Alapértelmezett felhasználó: **root** (user) és - (password)

Szerkesztő: **MySQL Workbench**

Környezeti változók:

- **MYSQL_HOME** →`c:\ProgramFiles\MySQL\5.7\bin`
- **Path** módosítása → `%Path%;%MYSQL_HOME%\bin`

Népszerű eszköz

A PHP-s webfejlesztés világában a MySQL szinte nélkülözhetetlen, szinte nincs olyan Shared Hosting szolgáltatás, ahol ne a MySQL lenne elérhető mint perzisztens réteg. Különféle engine-eket támogat, mindegyiknek meg(lehet) a maga előnye/-hátránya. Az InnoDB támogat professzionális használatot (hibái ellenére), viszont pl. *role*-okat ez sem kezel még.

BookStore db scriptek

Adatbázis, Szerep és Felhasználó létrehozása

[gradle|maven]\jboss\bookstore\database

```
1 CREATE DATABASE bookstoredb;
```

create-database.sql

```
1 CREATE ROLE bookstore_role WITH NOSUPERUSER NOCREATEDB
      NOCREATEROLE;
```

create-role.sql

```
1 CREATE USER bookstore_user;
2 ALTER USER bookstore_user PASSWORD '123topSECret321';
3 GRANT bookstore_role TO bookstore_user;
```

create-user.sql

BookStore db scriptek futtatása

Adatbázis, Szerep és Felhasználó létrehozása

```
1 > psql -d postgres -h localhost -p 5432 -U postgres -f
    create-database.sql
2
3 > psql -d postgres -h localhost -p 5432 -U postgres -f
    create-role.sql
4
5 > psql -d postgres -h localhost -p 5432 -U postgres -f
    create-user.sql
```

Csatlakozás a **bookstoredb** adatbázishoz a **bookstore_user** felhasználóval

```
1 > psql -d bookstoredb -h localhost -p 5432 -U bookstore_user
```

BookStore db schema

Könyv és kategóriája

```
1  CREATE TABLE bookcategory (
2    bookcategory_id INTEGER NOT NULL,
3    bookcategory_name CHARACTER VARYING(100) NOT NULL,
4    CONSTRAINT PK_BOOKCATEGORY_ID PRIMARY KEY (bookcategory_id)
5  );
6
7  ALTER TABLE bookcategory OWNER TO postgres;
8
9  CREATE TABLE book (
10   book_id SERIAL NOT NULL,
11   book_isbn CHARACTER VARYING(100) NOT NULL,
12   book_author CHARACTER VARYING(100) NOT NULL,
13   book_title CHARACTER VARYING(100) NOT NULL,
14   book_bookcategory_id INTEGER NOT NULL,
15   book_price REAL NOT NULL,
16   book_number_of_pages INTEGER NOT NULL,
17   CONSTRAINT PK_BOOK_ID PRIMARY KEY (book_id),
18   CONSTRAINT FK_BOOK_BOOKCATEGORY FOREIGN KEY (book_bookcategory_id)
19     REFERENCES bookcategory (bookcategory_id) MATCH SIMPLE ON UPDATE RESTRICT ON DELETE
         RESTRICT
20 );
21
22 ALTER TABLE book OWNER TO postgres;
23
24 CREATE UNIQUE INDEX UI_BOOK_ISBN ON book USING btree (book_isbn);
```

SERIAL: létre fog jönni egy book_book_id_seq nevű db sequence.

create-schema.sql

BookStore - grant

adatbázis konfiguráció utolsó lépései

```
1 GRANT SELECT, INSERT, UPDATE, DELETE ON bookcategory, book TO
      bookstore_role;
2
3 GRANT USAGE, SELECT, UPDATE ON book_book_id_seq TO bookstore_role;
```

grant-access.sql

```
1 > psql -d bookstoredb -h localhost -p 5432 -U postgres -f
      create-schema.sql
2
3 > psql -d bookstoredb -h localhost -p 5432 -U postgres -f
      grant-access.sql
```

BookStore - Adatfeltöltés

Kezdeti adatok létrehozása

```sql
INSERT INTO bookcategory (bookcategory_id, bookcategory_name)
    VALUES (0, 'SCIFI');
INSERT INTO bookcategory (bookcategory_id, bookcategory_name)
    VALUES (1, 'LITERATURE');
INSERT INTO bookcategory (bookcategory_id, bookcategory_name)
    VALUES (2, 'HISTORICAL');
INSERT INTO bookcategory (bookcategory_id, bookcategory_name)
    VALUES (3, 'PHILOSOPHY');

INSERT INTO book (book_isbn, book_author, book_title,
    book_bookcategory_id, book_price, book_number_of_pages)
    VALUES ('978-0441172719', 'Frank Herbert', 'Dune', 0, 3500,
    896);
[..]
```

initial-data.sql

```
> psql -d bookstoredb -h localhost -p 5432 -U postgres -f
    initial-data.sql
```

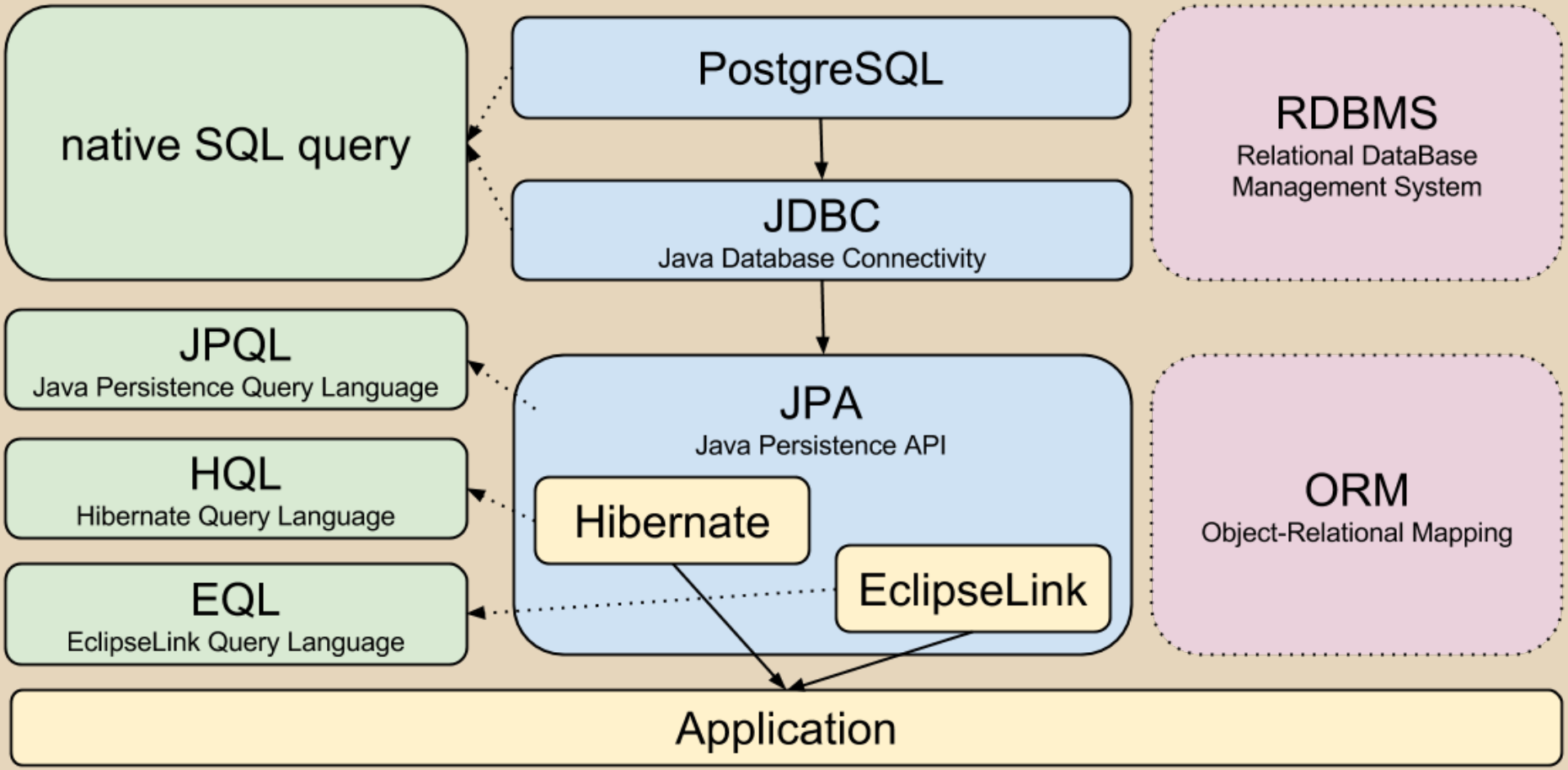
JDBC

Java SE témakörében ez gyakorlatban a `java.sql` csomag osztályainak használatát jelenti. Ezen kívül szükség lesz még a használni kívánt adatbázis **JDBC driver**-ére (**Java DataBase Connectivity**, tipikusan egy `jar` file, mely az ODBC (Open Database Connectivity) driver Java-s változata.

```
1  Class.forName("org.postgresql.Driver");
2  Connection connection =
       DriverManager.getConnection("jdbc:postgresql://localhost:5432/boo
       "bookstore_user", "123topSECret321");
3
4  Statement statement = connection.createStatement();
5  ResultSet rs = statement.executeQuery("SELECT book_id, book_title
       FROM book");
6  while (rs.next()) {
7    long id = rs.getLong("book_id");
8    String title = rs.getString("book_title");
9  }
10 rs.close();
11 statement.close();
12
13 connection.close();
```

A példakód a hibakezelést teljesen mellőzi. Működő példa a shopping projektben megtalálható, de a JPA használatához ennek előzetes megismerése nem szükséges. A JDBC ily használatának legnagyobb hátránya a **nem type-safe** kezelés.

Java Persistence API



- **Új module** létrehozása: **PostgreSQL JDBC driver**
- **Module regisztrálása** a `standalone domain`-hez
- **Database driver regisztrálása** a `standalone domain`-ben
- **bookstoredb datasource létrehozása** a `standalone domain`-ben

environment\modules\org\postgresql\main\

Könyvtár struktúra

[JBOSS_HOME]/modules/
- org/ → *name directory*
  - postgresql/ → *name directory*
    - main/ → *slot directory*
      - module.xml → *descriptor*
      - postgresql-42.1.4.jar → *JDBC driver* https://jdbc.postgresql.org/

```
1 <module xmlns="urn:jboss:module:1.1" name="org.postgresql">
2     <resources>
3         <resource-root path="postgresql-42.1.4.jar"/>
4     </resources>
5   <dependencies>
6     <module name="javax.api"/>
7     <module name="javax.transaction.api"/>
8   </dependencies>
9 </module>
```

module.xml

[JBOSS_HOME] | standalone | configuration | standalone.xml

```
1 <subsystem xmlns="urn:jboss:domain:ee:1.2">
2   <global-modules>
3     <module name="org.postgresql" slot="main"/>
4   </global-modules>
5   [..]
6 </subsystem>
```

standalone.xml

JBoss - Database driver regisztrálása

Standalone domain

[JBOSS_HOME] | standalone | configuration | standalone.xml

```
1  <subsystem xmlns="urn:jboss:domain:datasources:1.2">
2    <datasources>
3      [..]
4      <drivers>
5        [..]
6        <driver name="postgresql" module="org.postgresql">
7          <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-dat
8        </driver>
9      </drivers>
10   </datasources>
11 </subsystem>
```

standalone.xml

JBoss - Datasource létrehozása

PostgreSQL bookstoredb adatbázisa számára

[JBOSS_HOME] | standalone | configuration | standalone.xml

```
1  <subsystem xmlns="urn:jboss:domain:datasources:1.2">
2    <datasources>
3      [..]
4      <datasource jndi-name="java:jboss/datasources/bookstoreds"
           pool-name="BookStoreDSPool" enabled="true" use-java-context="true">
5        <connection-url>jdbc:postgresql://localhost:5432/bookstoredb</connection
6        <driver>postgresql</driver>
7        <security>
8          <user-name>bookstore_user</user-name>
9          <password>123topSECret321</password>
10       </security>
11     </datasource>
12     [..]
13   </datasources>
14 </subsystem>
```

standalone.xml

JNDI name: java:jboss/datasources/bookstoreds
JDBC url: jdbc:postgresql://localhost:5432/bookstoredb

JBoss - DB restart utáni Connection validáció

BookStore datasource konfiguráció módosítása

[JBOSS_HOME] | standalone | configuration | standalone.xml

```
1  <subsystem xmlns="urn:jboss:domain:datasources:1.2">
2    <datasources>
3      <datasource jndi-name="java:jboss/datasources/bookstoreds"
           pool-name="BookStoreDSPool" enabled="true" use-java-context="true">
4        [..]
5        <validation>
6          <check-valid-connection-sql>SELECT 1</check-valid-connection-sql>
7          <validate-on-match>true</validate-on-match>
8          <background-validation>false</background-validation>
9        </validation>
10       <statement>
11         <share-prepared-statements>false</share-prepared-statements>
12       </statement>
13     </datasource>
14     [..]
15   </datasources>
16 </subsystem>
```

standalone.xml

WebLogic DataSource konfigurálása

Server Administration Console-on keresztül

- PostgreSQL JDBC library hozzáadása a domain classpath-ához
  - JDBC driver (pl. `postgresql-9.4-1201.jdbc41.jar`) másolása ide: `[MW\_HOME]\user\_projects\domains\mydomain\lib\`
  - Server újraindítása
- WebLogic Server Administration Console
  - Services | Data Sources | Configuration tab
    - New → Generic Data Source
      * Name: **bookstoreds**
      * JNDI name: `jdbc/datasource/bookstoreds`
      * Target: "myserver"
      * Database type: PostgreSQL
    - Next, Next..
      * Database name: **bookstoredb**
      * Host name: **localhost**
      * Port: **5432** (default)
      * Database user name: **bookstore_user**
      * Password: **123topSECret321**
    - Érdemes tesztelni a kapcsolatot a felületen keresztül

Könyvtár struktúra

```
bookstore/
   [..]
   bs-persistence/
      src/
         main/
            java/
            resources/
               META-INF/
                  ejb-jar.xml
                  persistence.xml
      build.gradle
[..]
```

Root EAR project

Módosítás

```
1 [..]
2 dependencies {
3   [..]
4   deploy project('bs-persistence')
5 }
```

build.gradle

```
1 [..]
2 include 'bs-persistence'
```

settings.gradle

```
1 jar { archiveName 'bs-persistence.jar' }
```

build.gradle

JPA implementáció

JBoss 6.4 runtime a **Hibernate 4.3.10.Final**-t használja mint JPA implementáció, de mi kizárólag a JPA API-t (szeretnénk) használni fejlesztés közben, ezért nincs szükségünk erre a függőségre:
`'org.hibernate:hibernate-core:4.3.10.Final'`

EJB Service subproject

Persistence függőség beállítása

```
1 [..]
2 dependencies {
3   compile project(':bs-persistence')
4 }
```

build.gradle

Függőségek

`bs-weblayer` használja `bs-ejbservice`-t, hogy elérje a `BookFacade` EJB-t.

`bs-ejbservice` használja `bs-persistence`-t hogy elérje a `BookService` EJB-t.

hibernate.dialect: megadása nem kötelező, de ajánlott. Segítségével a Hibernate képes lesz PostgreSQL specifikus natív query-ket is generálni. Ha itt nem adjuk meg, akkor is működni fog a project (a generált ANSI SQL jelen esetben elegendő).

src | main | resources | META-INF | persistence.xml

```
1  <persistence xmlns="http://java.sun.com/xml/ns/persistence"
2           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3           xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
               http://java.sun.com/xml/ns/persistence/persistence_2_0.xs
4           version="2.0">
5    <persistence-unit name="bs-persistence-unit"
        transaction-type="JTA">
6      <jta-data-source>java:jboss/datasources/bookstoreds</jta-data-so
7      <properties>
8        <property name="hibernate.dialect"
             value="org.hibernate.dialect.PostgreSQLDialect"/>
9        <property name="hibernate.show_sql" value="true"/>
10       <property name="hibernate.format_sql" value="true"/>
11     </properties>
12   </persistence-unit>
13 </persistence>
```

Persistence unit neve: bs-persistence-unit
JNDI name: java:jboss/datasources/bookstoreds

JPA Persistence

Deployment descriptor

src | main | resources | META-INF | persistence.xml

```xml
<?xml version="1.0"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
        http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
    version="2.1">
  <persistence-unit name="bs-persistence-unit"
      transaction-type="JTA">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provi
    <jta-data-source>jdbc/datasource/bookstoreds</jta-data-source>
    <class>hu.qwaevisz.bookstore.persistence.entity.Book</class>
    <properties>
      <property name="eclipselink.logging.level" value="FINE"/>
    </properties>
  </persistence-unit>
</persistence>
```

Az Oracle WebLogic a **EclipseLink**-et (TopLink) használja mint JPA Provider, és a Hibernate beállítása csak úgy lehetséges, ha kicseréljük minden függőségével együtt az elemeket (nem javasolt). Viszont a JPA Provider-ek között is szabadon mozoghatunk a JavaEE világában, egyedül a descriptorokat kell cserélnünk/finomhangolnunk.

Book entitás

Identification mező

Ha nem adjuk meg a `@SequenceGenerator` és `@GeneratedValue` annotációkat, beszúráskor egy `org.hibernate.id.IdentifierGenerationException` fogunk kapni, melyben jelzi a Hibernate hogy: `"ids for this class must be manually assigned before calling save()"`.

```
1  package hu.qwaevisz.persistence.entity;
2
3  @Entity
4  @Table(name = "book")
5  public class Book implements Serializable {
6
7    @Id
8    @SequenceGenerator(name = "generatorBook", sequenceName =
         "book_book_id_seq", allocationSize = 1)
9    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
         "generatorBook")
10   @Column(name = "book_id", nullable = false)
11   private Long id;
12
13   [..]
14 }
```

Database **table name**: book

Database **sequence name**: book_book_id_seq

Database **column name**: book_id

Book.java

Book entitás

További mezők

```java
@Column(name = "book_isbn", nullable = false)
private String isbn;

@Column(name = "book_author", nullable = false)
private String author;

@Column(name = "book_title", nullable = false)
private String title;


@Enumerated(EnumType.ORDINAL)
@Column(name = "book_bookcategory_id", nullable = false)
private BookCategory category;

@Column(name = "book_price", nullable = false)
private Double price;

@Column(name = "book_number_of_pages", nullable = false)
private Integer numberOfPages;
```

A **bookcategory** adatbázis szinten egy tábla, ORM szinten viszont egy `enum` (CRUD műveletek nem értelmezettek rajta).

Book.java

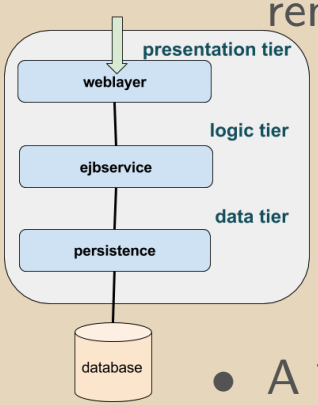
BookCategory

ORM által használt enum

```java
package hu.qwaevisz.bookstore.persistence.entity.trunk;

public enum BookCategory {

  SCIFI,
  LITERATURE,
  HISTORICAL,
  PHILOSOPHY;

}
```

BookCategory.java

- A **BookStore** alkalmazás két szinten definiál szolgáltatásokat.
  - A `bs-ejbservice` project szintjén definiált EJB service-ek a "customer" (megrendelő) igényei szerint alakítandóak.



    - Az interface által megfogalmazott igényekre üzleti *use-case*-ek születnek.
    - Az interface által használt típusok a **stub**-ok, melyek technikai szinten publikusan kifelé nézve.
    - Az interface által használt kivételek technikai szinten publikusak, vagy publikus hibaüzenetekké alakíthatóak. Technikai részleteket ritkán tartalmaznak (security).
  - A `bs-persistence` project szintjén definiált EJB service-ek a fejlesztők belső szabályai alapján, elsősorban technikai követelmények és lehetőségek szerint alakulnak.
    - Az interface által megfogalmazott igényeket a fejlesztők maguk alakítják ki, egészséges egyensúlyt tartva a redundancia mentes, karbantartható, tiszta és olykor akár future-proof követelmények alapján.
    - Az interface által használt típusok az **entity**-k, melyek az alkalmazás belső használatú DTO-i, kifelé legtöbbször nem publikusak.
    - Az interface által használt kivételek részletes technikai információkat is tartalmazhatnak, legtöbbször ezek a "kliens" oldalon nem jelennek meg és nem alakíthatóak közvetlenül publikus hibaüzenetté.

BookService - Persistence SLSB

Local interface

```java
package hu.qwaevisz.bookstore.persistence.service;
[..]
@Local
public interface BookService {
  [..]
  Book read(String isbn) throws PersistenceServiceException;

  List<Book> readAll() throws PersistenceServiceException;
  [..]
}
```

BookService.java

BookService - Persistence SLSB

Implementáció

```
1  @Stateless(mappedName = "ejb/bookService")
2  @TransactionManagement(TransactionManagementType.CONTAINER)
3  @TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
4  public class BookServiceImpl implements BookService {
5
6    @PersistenceContext(unitName = "bs-persistence-unit")
7    private EntityManager entityManager;
8
9    @Override
10   public Book read(final String isbn) throws PersistenceServiceException {
11     Book result = null;
12     try {
13       result = this.entityManager.createNamedQuery(Book.GET_BY_ISBN,
               Book.class).setParameter("isbn", isbn).getSingleResult();
14     } catch (final Exception e) {
15       throw new PersistenceServiceException("Unknown error when fetching Book by ISBN
               (" + isbn + ")! " + e.getLocalizedMessage(), e);
16     }
17     return result;
18   }
19   [...]
20 }
```

JPA **Named Query**: `Book.GET_BY_ISBN` (`"Book.getByIsbn"`)
Paraméter neve: `isbn`
`getSingleResult()`: pontosan egy rekord létezik (különben kivételt dob)
`getResultList()`: `List<Book>` lesz a visszatérési érték

JPA Named Query

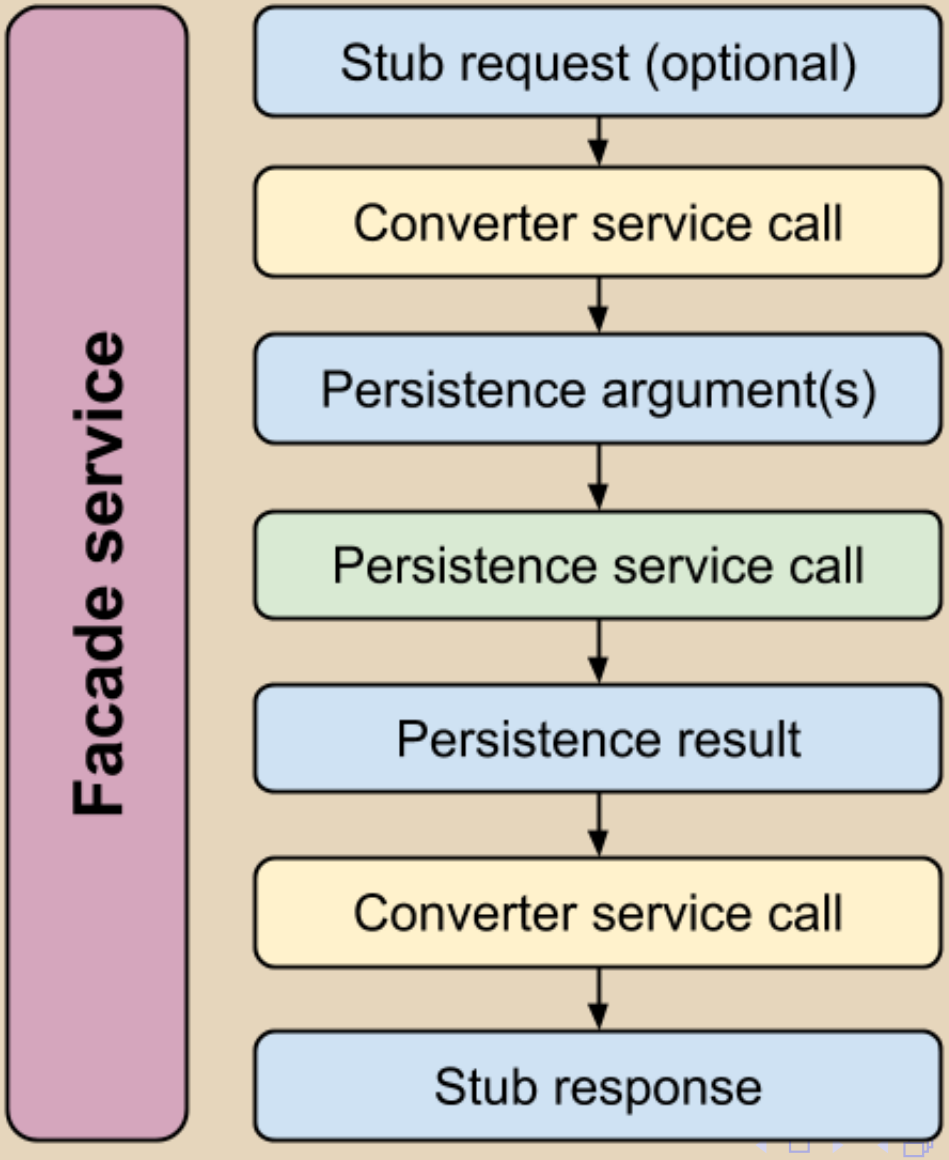
JPQL syntax

```
1  @Entity
2  @Table(name = "book")
3  @NamedQueries(value = { //
4      @NamedQuery(name = Book.GET_BY_ISBN, query = "SELECT b FROM
           Book b WHERE b.isbn=:isbn"),
5      @NamedQuery(name = Book.GET_ALL, query = "SELECT b FROM Book
           b ORDER BY b.title"),
6      [..]
7  })
8  public class Book implements Serializable {
9
10   public static final String GET_BY_ISBN = "Book.getByIsbn";
11   public static final String GET_ALL = "Book.getAll";
12
13   [..]
14 }
```

Book.java

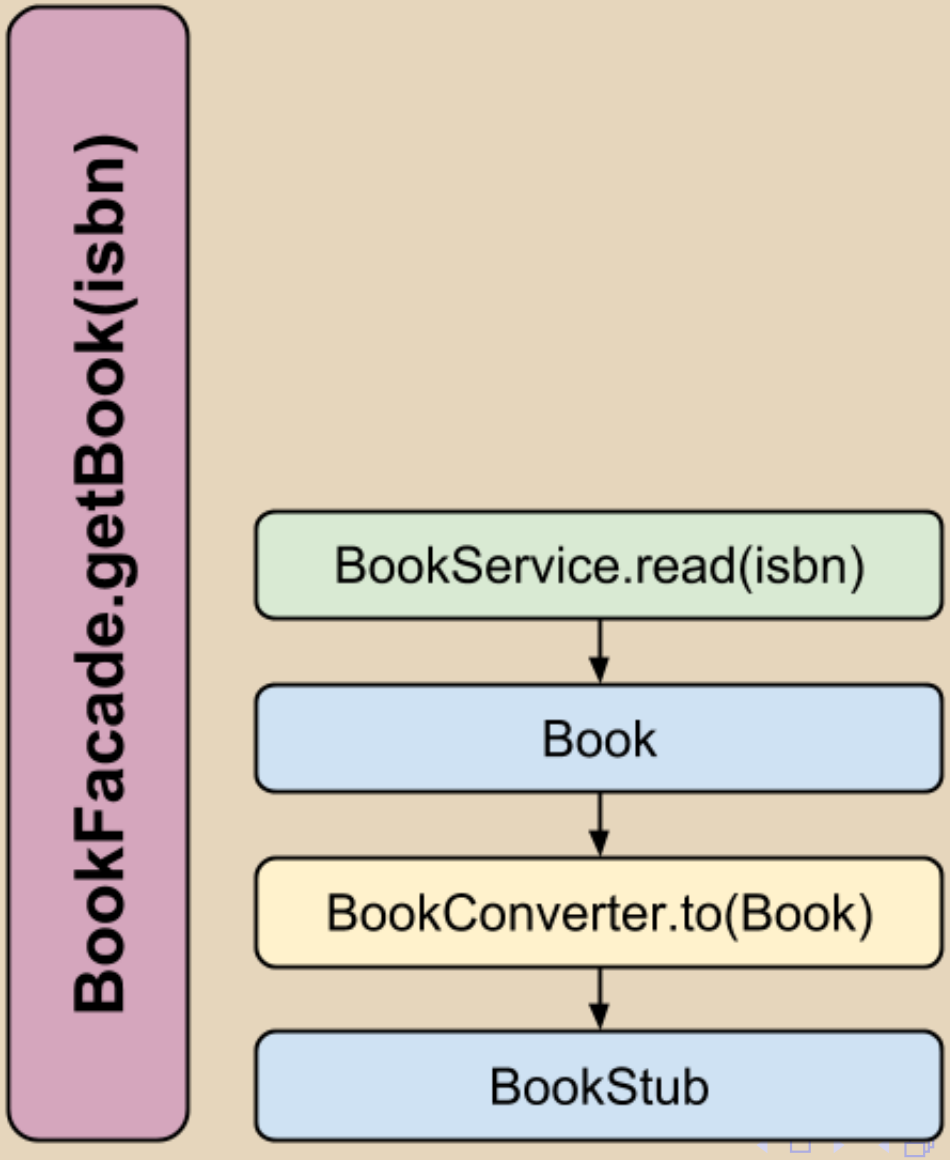
Facade réteg felelőssége

EJB Service project (`bs-ejbservice`)



Könyv lekérdezése ISBN alapján

EJB Service project (`bs-ejbservice`)



BookConverter

EJB Service project (bs-ejbservice)

```java
@Local
public interface BookConverter {
  BookStub to(Book book);
  List<BookStub> to(List<Book> books);
}
```

BookConverter.java

Nagyon jó 3rd party library-k léteznek konverziós célra, pl. **MapStruct** (http://mapstruct.org/).

```java
@Stateless
public class BookConverterImpl implements BookConverter {

  @Override
  public BookStub to(Book book) {
    final BookCategoryStub category =
        BookCategoryStub.valueOf(book.getCategory().toString());
    return new BookStub(book.getIsbn(), book.getAuthor(), book.getTitle(), category,
        book.getPrice(), book.getNumberOfPages());
  }
  [..]
}
```

BookConverterImpl.java

BookFacade módosítása

EJB Service project (bs-ejbservice)

```
1  @Stateless(mappedName = "ejb/bookFacade")
2  public class BookFacadeImpl implements BookFacade {
3
4    @EJB
5    private BookService service;
6
7    @EJB
8    private BookConverter converter;
9
10   @Override
11   public BookStub getBook(String isbn) throws FacadeException {
12     try {
13       final BookStub stub = this.converter.to(this.service.read(isbn));
14       if (LOGGER.isDebugEnabled()) {
15         LOGGER.debug("Get Book by isbn (" + isbn + ") --> " + stub);
16       }
17       return stub;
18     } catch (final PersistenceServiceException e) {
19       LOGGER.error(e, e);
20       throw new FacadeException(e.getLocalizedMessage());
21     }
22   }
23   [..]
24 }
```

BookFacadeImpl.java

Megjelenítési réteg továbbfejlesztése

bs-weblayer project

A korábbi `BookPingServlet` mellett készítsünk egy valós megjelenítési réteget Servlet/JSP alapokon, követve a *Modell-View-Controller* (MVC)tervezést.

- **BookController** servlet, mely URI paraméterben kapja az ISBN számot, mely alapján lekéri az érintett könyvet, majd a `BookStub` példányt elhelyezi a `HttpServletRequest`-ben, és a kérést *forward*-olja egy *JSP* lapnak.
- **book.jsp** által generált servlet, mely kiveszi a `HttpServletRequest`-ből a `BookStub` példányát és megjeleníti XHTML tagek ölelésében.

BookController

Weblayer project (bs-weblayer)

```
1  package hu.qwaevisz.bookstore.weblayer.servlet;
2  [..]
3  @WebServlet("/Book")
4  public class BookController extends HttpServlet {
5    [..]
6    @EJB
7    private BookFacade facade;
8
9    @Override
10   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
         ServletException, IOException {
11     String isbn = request.getParameter("isbn");
12     [..]
13     try {
14       BookStub book = this.facade.getBook(isbn);
15       request.setAttribute("book", book);
16     } catch (final FacadeException e) {
17       LOGGER.error(e, e);
18     }
19     RequestDispatcher view = request.getRequestDispatcher("book.jsp");
20     view.forward(request, response);
21   }
22   [..]
23 }
```

BookController.java

book.jsp

Weblayer project (bs-weblayer)

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
2  <%@ page import="hu.qwaevisz.bookstore.ejbservice.domain.BookStub" %>
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7  <title>:: Book ::</title>
8  </head>
9  <body>
10   <jsp:useBean id="book" class="hu.qwaevisz.bookstore.ejbservice.domain.BookStub"
         scope="request" />
11   <h1>
12     <jsp:getProperty name="book" property="author" />:
13     <jsp:getProperty name="book" property="title" />
14   </h1>
15   <div>
16     <label>ISBN: </label>
17     <span><jsp:getProperty name="book" property="isbn" /></span>
18   </div>
19   <div>
20     <label>Number of pages: </label>
21     <span><jsp:getProperty name="book" property="numberOfPages" /></span>
22   </div>
23   [..]
24 </body>
25 </html>
```

book.jsp

`http://localhost:8080/bs-weblayer/Book?isbn=978-0441172719`

További feladatok:

- Összes könyv listázása (JPA query, persistence service, facade service, converter ejb)
  - Criteria alapján szűrés megvalósítása (opcionális)
- `list.jsp` és `BookListController` servlet megvalósítása
- `list.jsp` és `book.jsp` összekötése (a controller servlet-ek összekötése)
- `page.css` file készítése a weblapokhoz
- ...

A **BookStore** alkalmazás a lekérdezési lehetőségek mellett teljes CRUD szerkesztési lehetőséget is tartalmaz. Ennek technikai részleteit JPA szempontjából később tekintjük át.

A következő néhány slide-on - csupán előzetesen - az érintett műveletek persistence oldala lesz olvasható. A megoldás nélkülözi a tranzakció- és hibakezelést. Mindezek később a **school** elnevezésű projektben lesznek részletezve.

```
1  @Stateless(mappedName = "ejb/bookService")
2  public class BookServiceImpl implements BookService {
3    [..]
4    @PersistenceContext(unitName = "bs-persistence-unit")
5    private EntityManager entityManager;
6
7    @Override
8    public Book create(final String isbn, final String author, final String title, final
         int numberOfPages, final double price, final BookCategory category)
9        throws PersistenceServiceException {
10     [..]
11     final Book book = new Book(isbn, author, title, numberOfPages, price, category);
12     this.entityManager.persist(book);
13     [..]
14     return book;
15   }
16   [..]
17 }
```

BookServiceImpl.java

```java
@Stateless(mappedName = "ejb/bookService")
public class BookServiceImpl implements BookService {
  [..]
  @PersistenceContext(unitName = "bs-persistence-unit")
  private EntityManager entityManager;

  @Override
  public Book update(final String isbn, final String author, final String title, final
       int numberOfPages, final double price, final BookCategory category)
      throws PersistenceServiceException {
    final Book book = this.read(isbn);
    book.setAuthor(author);
    book.setTitle(title);
    book.setNumberOfPages(numberOfPages);
    book.setPrice(price);
    book.setCategory(category);
    [..]
    return this.entityManager.merge(book);
  }
  [..]
}
```

BookServiceImpl.java

```
1  @Stateless(mappedName = "ejb/bookService")
2  public class BookServiceImpl implements BookService {
3    [..]
4    @PersistenceContext(unitName = "bs-persistence-unit")
5    private EntityManager entityManager;
6
7    @Override
8    public void delete(final String isbn) throws PersistenceServiceException {
9      this.entityManager.createNamedQuery(Book.REMOVE_BY_ISBN).setParameter("isbn",
           isbn).executeUpdate();
10     [..]
11   }
12   [..]
13 }
```

BookServiceImpl.java

A hivatkozott **JPQL** az alábbi:

```
DELETE FROM Book b WHERE b.isbn=:isbn
```

A kódolás - mint fejlesztési folyamat - szét nem választható egységet képez a teszteléssel. Jelen diasorozatnak azonban nem scope-ja a kódolást a minden ponton vele együtt járó teszteléssel együtt bemutatni.

- **egység** vagy komponens- teszt: az objektum-orientált fejlesztésben egy osztály (egy felelősség) önálló, független tesztjét értjük egység teszt alatt. Minden környezeti és közel minden programozott függőséget (elsősorban az aggregációkat és kompozíciókat) ki kell küszöbölni, meg kell kerülni. Célja legtöbbször a programhibák leggyorsabb és legpontosabb felderítése, illetve relatíve magas *coverage* esetén a refactorálás támogatása.
- **integrációs** teszt: néhány komponens együttes tesztelése, célja legtöbbször a fejlesztés menetének meggyorsítása. Mindig helyben dönthető el, hogy mekkora környezeti függőséget engedélyezünk a teszt számára.
- **system** vagy rendszer- teszt: tipikusan a fejlesztők által végzett legmagasabb tesztelés, melyben a rendszer vagy annak egy önállóan működő része "minden" függőségével együtt kerül tesztelésre (itt képke kerülhetnek szimulátorok, melyeket szintjén a fejlesztőknek kell lefejlesztenie).
- **end-to-end** vagy átvételi- teszt: egy jól működő projektben az end-to-end tesztelést külön, a fejlesztőktől független csapat végzi. Mérnökökből álló tesztelői (integrációs és verifikációs) csapat automata tesztekben gondolkodik, míg sok helyen végzik ezt a szintet manuális lépések segítségével. Webalkalmazás fejlesztés során gyakoriak pl. a *Selenium* tesztek.

Perzisztencia tesztelése

- Körülményes egy-egy named query tesztelése miatt deploy-olni és manuális tesztelni (nagyon sok idő, sok elgépelési lehetőség).
- Adatbázis műveletek, `named query`-k tesztelése nem unit-test felelősség.
- A JPA API használható *standalone* alkalmazásban is. Ez is egy mód a tesztelésre, de ez egy elkülönített kódrész lesz, mely automata tesztelésre legtöbbször nem alkalmas (de a fejlesztés támogatására igen).
- Talán a legelegánsabb megoldás **integration test**-eket készíteni a persistence réteg tesztelésére, melyben a megírt named query-k és entity manager műveletek valóban lefutnak az adatbázison, azok hatékonysága és teljesítménye mérhető és ellenőrizhető lesz, függetlenül az alkalmazás szervertől (az alkalmazás szervert kivesszük a tesztelésből, de az adatbázist nem).
- Az integrációs tesztek tipikusan nem futnak az egység tesztekkel együtt, így a külön futásukat illendő megoldani (többféle megoldás lehetséges).
- A persistence réteg osztályait (ahol értelme van, és nem boilerplate algoritmusok érintettek) természetesen egység tesztekkel le lehet fedni. Az egység tesztelésről a **school** elnevezésű projekt fog részletesebben foglalkozni.

Standalone Persistence Unit az integrációs tesztekhez

src | main | resources | META-INF | persistence.xml

```
1  <persistence [..]>
2    [..]
3    <persistence-unit name="bs-persistence-test-unit" transaction-type="RESOURCE_LOCAL">
4      <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
5      <class>hu.qwaevisz.bookstore.persistence.entity.Book</class>
6      <properties>
7        <property name="javax.persistence.jdbc.url"
               value="jdbc:postgresql://localhost:5432/bookstoredb" />
8        <property name="javax.persistence.jdbc.user" value="bookstore_user" />
9        <property name="javax.persistence.jdbc.password" value="123topSECret321" />
10       <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
11
12       <property name="hibernate.dialect"
               value="org.hibernate.dialect.PostgreSQLDialect"/>
13       <property name="hibernate.show_sql" value="true"/>
14       <property name="hibernate.format_sql" value="true"/>
15
16       <prope
17     </proper
18   </persiste
19 </persistenc
```

A JPA a classpath fix helyén keresi a `persistence.xml`-t. E miatt a `jboss.as.jpa.managed` értékét `false`-ra állítva elérhetjük hogy runtime a JBoss ne foglalkozzon ezzel a standalone `persistence unit`-tal (ClassNotFound hibát okoz e nélkül). JNDI név helyett most definiálva van a **provider** class valamint a csatlakozás adatai. Mivel nincs container aki átnézné a kódot annotációk után, an entity-ket fel kell sorolni.
Persistence unit neve: `bs-persistence-test-unit`

Integration teszt classpath

Az integrációs tesztek JavaEE container nélkül futnak (pl. gradle/maven/eclipse JVM-je futtatja őket), így nem elégséges csupán a interface-ek/enum-ok halmazát tartalmazó függőség beállítása. Érdemes a futtató környezet által használt runtime függőségeket felsorolni, de a JavaEE szabványos világában akár más runtime környezetet is felépíthetünk.

Szükséges runtime függőségek:

- **PostgreSQL**
  - `org.postgresql:postgresql:9.4+`
- **Hibernate**
  - `org.hibernate:hibernate-core:4.3.10.Final`
  - `org.hibernate:hibernate-entitymanager:4.3.10.Final`
- **JavaEE 6.0 implementáció**
  - `org.jboss.spec:jboss-javaee-6.0:3.0.3.Final`
  - **Fontos!** Ezzel egy időben nem szabad a classpath-ra tenni a JavaEE 6.0 API-t (a teszt classpath része a production classpath):
    - `javax:javaee-api:6.0`

Root project - Alprojectek megkülönböztetése

JavaEE 6.0 API leszedése a `compile` classpath-ról

`configure(..)`: Az összes alprojektre érvényes, kivéve a `bs-persistence` nevűre, ahol a `testCompile`-ra felkerül majd a Java EE 6.0 implementáció.

```
1  [..]
2  allprojects {
3    repositories {
4      [..]
5    }
6  }
7  subprojects {
8    [..]
9    dependencies {
10     compile group: 'log4j', name: 'log4j', version: log4jVersion
11     testCompile group: 'org.testng', name: 'testng', version:
           testngVersion
12   }
13   test {
14     useTestNG()
15   }
16 }
17 configure(subprojects.findAll {it.name != 'bs-persistence'}) {
18   dependencies {
19     compile group: 'javax', name: 'javaee-api', version: jeeVersion
20   }
21 }
22 [..]
```

build.gradle

Persistence project - Teszt classpath konfigurálása

JavaEE 6.0 implementáció elhelyezése a `testCompile` classpath-ra

```
1  [..]
2  dependencies {
3    compileOnly group: 'javax', name: 'javaee-api', version: jeeVersion
4
5    testCompile group: 'org.postgresql', name: 'postgresql', version:
         postgresqlVersion
6    testCompile group: 'org.hibernate', name: 'hibernate-core', version:
         hibernateVersion
7    testCompile group: 'org.hibernate', name: 'hibernate-entitymanager',
         version: hibernateVersion
8    testCompile group: 'org.jboss.spec', name: 'jboss-javaee-6.0',
         version: jbossjee6Version
9  }
10 [..]
```

`build.gradle`

A **compileOnly** dependency configuration a *java plugin* része Gradle 2.12 óta. Így a JavaEE API nem lesz része a `testCompile`-nak, ami az integrációs teszteket elrontaná. Jó - bár talán kevésbé elegáns megoldás - a `compile` classpath-ra elhelyezni a JavaEE 6.0 implementációt.

src | test | 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

A részletes log nagyon sokat segít a fejlesztésben és a tesztelésben is. És pontosan ezen részletek miatt lesz nagyon gyors és hatékony pl. a named query-k optimalizálása. Mivel az integrációs tesztek futása során nincs JavaEE container aki biztosítja a naplózást, nekünk kell ezt is konfigurálnunk.

Integrációs teszt

Előkészületek

```
1  package hu.qwaevisz.bookstore.persistence.service;
2  [..]
3  public class BookServiceImplIntegrationTest {
4
5    private static final String PERSISTENCE_UNIT = "bs-persistence-test-unit";
6
7    private BookServiceImpl object;
8
9    @BeforeClass
10   private void setUp() {
11     final EntityManagerFactory factory =
           Persistence.createEntityManagerFactory(PERSISTENCE_UNIT);
12     final EntityManager entityManager = factory.createEntityManager();
13
14     this.object = new BookServiceImpl();
15     this.object.setEntityManager(entityManager);
16   }
17
18   [..]
19
20   @AfterClass
21   private void tearDown() {
22     this.object.getEntityManager().close();
23   }
24 }
```

A lokális `entityManager` "befecskendezésének" megkerülése miatt egy default/-protected `setEntitymanager()` mutator metódust készítenünk kell a production kódban :-(

`BookServiceImplIntegrationTest.java`

Integrációs teszt

A `read()` mögötti named query működésének tesztelése.

Az `integration` csoportosításra a unit-test-ektől való megkülönböztetés miatt van szükség.

```
1  public class BookServiceImplIntegrationTest {
2    [..]
3    @Test(groups = "integration")
4    private void readSampleBookFromDatabase() throws PersistenceServiceException {
5      final Book book = this.object.read("978-0441172719");
6      this.assertBook(book, "978-0441172719", "Frank Herbert", "Dune",
           BookCategory.SCIFI, 896, 3500d);
7    }
8
9    private void assertBook(final Book book, final String isbn, final String author,
         final String title, final BookCategory category,
10       final Integer numberOfPages, final double price) {
11     Assert.assertEquals(book.getIsbn(), isbn);
12     Assert.assertEquals(book.getAuthor(), author);
13     Assert.assertEquals(book.getTitle(), title);
14     Assert.assertEquals(book.getCategory(), category);
15     Assert.assertEquals(book.getNumberOfPages(), numberOfPages);
16     Assert.assertEquals(book.getPrice(), price);
17   }
18   [..]
19 }
```

BookServiceImplIntegrationTest.java

Integrációs teszt

Adatmanipuláció tranzakcióban

```
1  public class BookServiceImplIntegrationTest {
2    [..]
3    private static final String NEW_BOOK_ISBN = "42-NEW-BOOK-ISBN";
4
5    @Test(groups = "integration")
6    private void createABook() throws PersistenceServiceException {
7      if (this.object.exists(NEW_BOOK_ISBN)) {
8        this.object.getEntityManager().getTransaction().begin();
9        this.object.delete(NEW_BOOK_ISBN);
10       this.object.getEntityManager().getTransaction().commit();
11     }
12
13     this.object.getEntityManager().getTransaction().begin();
14     this.object.create(NEW_BOOK_ISBN, "Lorem Ipsum", "Sample book", 142, 999,
           BookCategory.HISTORICAL);
15     this.object.getEntityManager().getTransaction().commit();
16
17     final Book book = this.object.read(NEW_BOOK_ISBN);
18     this.assertBook(book, NEW_BOOK_ISBN, "Lorem Ipsum", "Sample book",
           BookCategory.HISTORICAL, 142, 999);
19   }
20   [..]
21 }
```

Itt most nincs EJB container, "ami" gondoskodna a tranzakciókról. Ezt nekünk kell megtennünk minden adatmanipulációs esetben! Illetve valamiképpen a *rerunable* képességről is gondoskodnunk kell.

TestNG Integration Suite

bs-persistence project

```
1 [..]
2 test {
3   useTestNG {
4     // suites 'src/test/resources/testng-unit.xml'
5     suites 'src/test/resources/testng-integration.xml'
6     // excludeGroups 'integration'
7     // includeGroups 'unit'
8   }
9 }
```

build.gradle

Többféle megoldás lehetséges az integration tesztek futtatására:

- Külön **integration project** létrehozása (talán a legelegánsabb)
- Külön konfigurációs **profil** létrehozása (külön classpath, testng config, stb.)
- Külön TestNG `suite.xml`-ek definiálása az egység ill. az integration tesztek számára (az egyszerűség kedvéért ezt alkalmazzuk)
- Globálisan kizárni/hozzáadni bizonyos TestNG group-okat a tesztek futásához

Integration TestNG suites

src | test | resources | testng-*.xml

```xml
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="integration-test-suite" verbose="1" >
  <test name="bookservice">
    <classes>
      <class name="hu.[..].service.BookServiceImplIntegrationTest" />
      <class name="hu.[..].service.BookSearchImplIntegrationTest" />
    </classes>
  </test>
</suite>
```

testng-integration.xml

```xml
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="unit-test-suite" verbose="1">
  <test name="bookservice">
    <groups>
      <run>
        <exclude name="integration" />
      </run>
    </groups>
    <packages>
      <package name="hu.[..].service.*" />
    </packages>
  </test>
</suite>
```

Az összes teszt case unit test ami a megadott package-ben megtalálható, kivéve az ami az integration csoportba tartozik.

testng-unit.xml