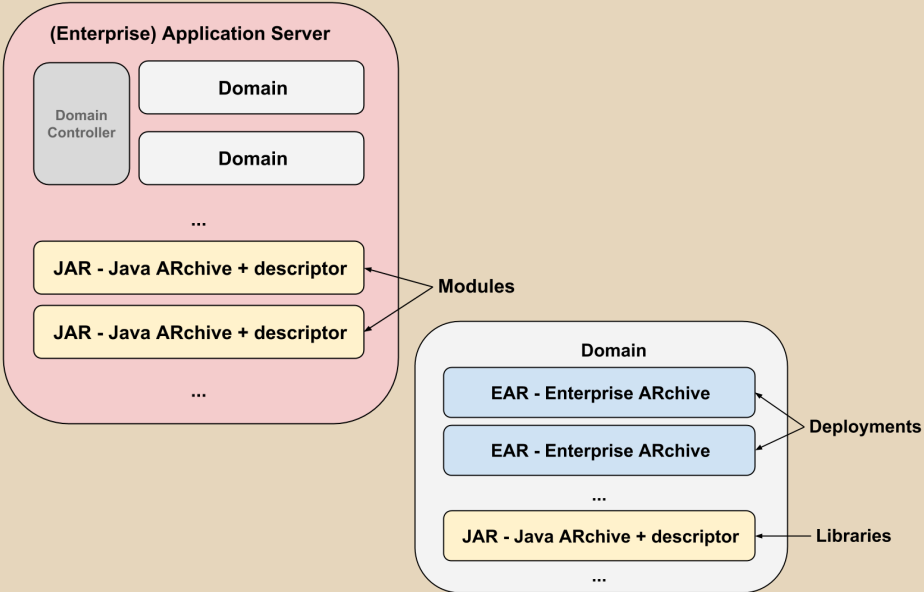# BookStore #gradle

Enterprise Application, Git, EJB, EAP/EAS, Logging, PostgreSQL/MySQL, JPA, Integration testing
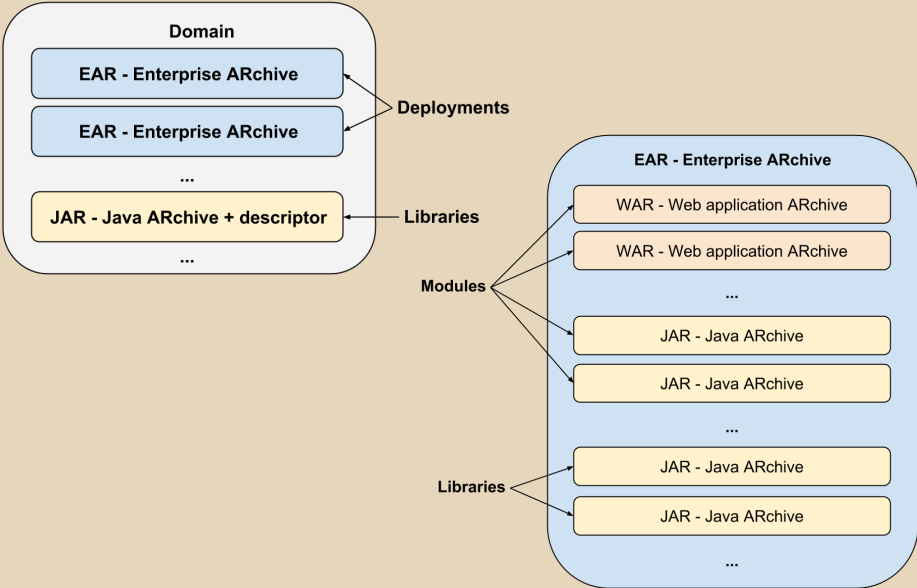
Óbuda University, Java Enterprise Edition

John von Neumann Faculty of Informatics

Lab 3

Dávid Bedők
2018-01-17
v1.1

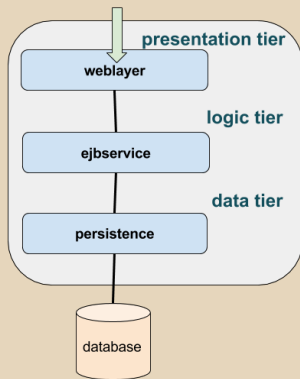# EAS - Domain structure

EAS - Deployment structure

Task: create an application which can list the books of a store..
We will implement the task in two parts. In the first round the persistence
layer will be mocked/faked.

Applicable techniques:

  ▷ 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

Test the development environment

```
1 > cd [GIT]\gradle\jboss\booklight\
2 > gradle clean build
3 ..
4 BUILD SUCCESSFUL
```

New EAR artifact:

`[GIT]\gradle\jboss\booklight\build\libs\booklight.ear`

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

Copy the compiled **booklight.ear** file into the
`[JBOSS-HOME]\standalone\deployments` directory.
http://localhost:8080/bl-weblayer/BookPing

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

▷ Create project hierarchy

▷ Basics of the **Enterprise JavaBean** (`ejb`) as **business layer**

▷ Simple servlet as **weblayer**

▷ Mocked data as **persistence layer**

Enterprise Application Servers (EAS)

▷ RedHat JBoss 6.4

▷ Oracle WebLogic 12.1.3

Possibility to choose another EAS

A JavaEE application can be run any complient JavaEE application server which support the used JavaEE version. This interoperability further strengthens the **cross-platform** behavior of the Java ecosystem.

We will deploy the task in two different EAS(s).

Cross-Platform Java
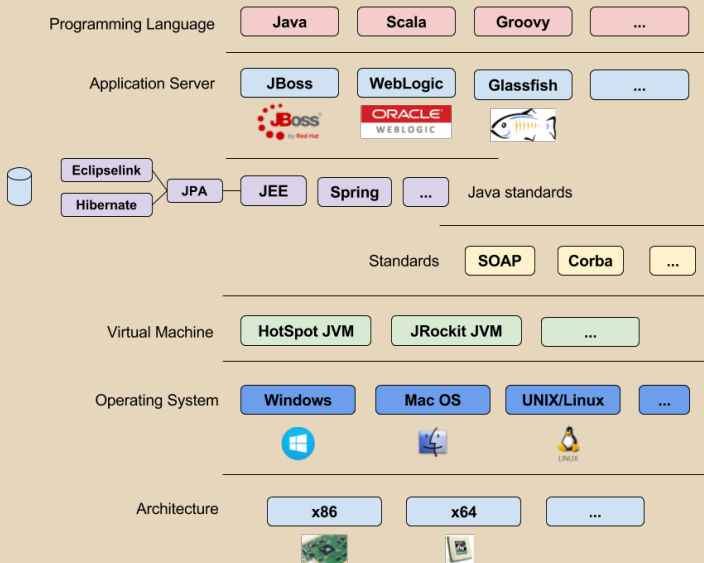
Repository

◈ [gradle|maven]\jboss\booklight

```
1 > gradle clean build
```

Note: the build of Gradle will execute **ear** in case of ear plugin, **war** in case of war plugin and **compile** in case of java plugin (these are all depend on the build).

Git basics
Discard local changes

Refreshing the blueprint repository (Git `pull`) could be necessary if the `origin/master` was changed. In case of that you have to discard all of your local changes.

Select working copy (in the root directory)

```
1 > git checkout .
```

Discard local changes

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

Discard new files

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

Get the new changes from **master**

```
1 > git pull
```

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

Version: **2.50.02**
OS: **Windows** (Mono 5.0 is required in case of NIX)

It provides graphic interface for the `Git` commands. Not only a tool, it teaches the commands as well.

Using the Git it not always a linear use-case, the same goal can be accessible many different ways. Because of this using any tool could carry some risks. There is no risk for basic use-cases.

https://www.sourcetreeapp.com/

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

Elegant, modern graphical interface. Sometimes it is a bit circumstantial (e.g. in case of git log/blame), but it could be an effective tool after all. It doesn't teach the git commands like the GitExtensions.

Git Commit and Push
Share new improvements and bug fixes

Check the status

```
1 > git status
```

Stage

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

Commit (-a/–all, -m/–message)

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

Send the new changes to the master

```
1 > git push
```

Multiple projects in hierarchy:

- ▷ 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

Directory structure

```
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 (subdirectories are also possible)

```xml
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

EAR - Enterprise ARchive

Directory structure

```
META-INF/
    MANIFEST.MF
    application.xml
lib/
    dummy.jar      3rd party library
xyz.jar
abc.war
```

```xml
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

Create multi-project directory structure
Simplified bookstore

Directory structure

```
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

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

</ejb-jar>
```

ejb-jar.xml

It is mandatory?

The deployment descriptor file of the EJB module is required for Maven even if this file is entirely empty (in a simple case we can describe almost everything via annotations, so the `ejb-jar.xml` file remains empty). The precedence of the descriptor file is always higher than the annotations.

Root EAR project
Gradle's EAR plugin

> Dependency configurations of the `ear plugin`: **deploy** and **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

```
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 }
```

This is exactly the default value, so the entire block can be omitted.

build.gradle

WebLayer subproject

dependency configuration of the **war** plugin:

▷ **providedCompile**: it needs for the compilation but the container provides it

▷ **providedRuntime**: the container provides it

We can use the **compile** dc of the **java** plugin as well: it needs for the compilation but the container does not provide it (it will be copied into the WEB-INF\lib of the *war* artifact).

```
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

Meaning of the ":" sign before the name of the project: *absolute* path instead of *relative*.

Enterprise JavaBeans

▷ Server side business components (back-end services)

▷ JSR19, JSR152, JSR220, JSR318, JSR345

▷ IBM (1997), later Sun Microsystems (1999)

▷ Kind of 'best-practice' in order to implement only the 'business value' instead of the typical 'decorations' (boilerplate codes).
  - Transaction management
  - Concurrency management
  - Asynchronous method call
  - Event management
  - ava Message Service integration (Message Driven Beans)
  - Persistence support integration (but the persistence is no longer part of the EJB specification)

▷ Verziók
  - EJB1.x: all "remote"
  - EJB2.x: very uncomfortable, overcomplicated
  - EJB3.x: re-thinking concept based on the experiences of the Spring framework and Hibernate (uses POJOs just like the Spring FW and breaks with the entity beans (supports but does not implement it))

EJB types

▷ **Session Beans**
- **Stateless SB** (SLSB)
  - ◦ Stateless
- **Stateful SB** (SFSB)
  - ◦ Statefull
  - ◦ Activation/Passivation
  - ◦ The serialization is important
- **Singleton SB** (SSB)
  - ◦ Singleton

▷ **Message Driven Beans** (MDB)
- Message driven
- Primarily asynchronous processing
- There is no client interface

▷ **Entity Beans**
- Deprecated, JPA substitutes from EJB 3.x

Business Layer

Define some business methods:

- ▷ Based on the ISBN[1] select a Book from the catalog
    - `BookStub getBook ( String isbn )`
- ▷ Based on same criteria list the books which correspond to the conditions
    - `List<BookStub> getBooks ( BookCriteria criteria )`
- ▷ `FacadeException`: when the request cannot be executed

Define the required DTO[2]s:

- ▷ 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 classes - Book categories

```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

```java
package hu.qwaevisz.booklight.ejbservice.domain;
public class BookStub {
  private String isbn;
  private String author;
  private String title;
  private BookCategoryStub category;
  private double price;
  private int numberOfPages;

  public BookStub(String isbn, String author, String title,
      BookCategoryStub category, double price, int numberOfPages)
      {
    this.isbn = isbn;
    this.author = author;
    this.title = title;
    this.category = category;
    this.price = price;
    this.numberOfPages = numberOfPages;
  }
  [...]
}
```

BookStub.java

Domain classes - Search criteria

```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 classes - Throwable business exception

```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

```java
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: expensive and slow (network usage, serialization), RMI, the original concept supported only this, currently we use it between different JVMs
@Local: cheap and fast, inside the same JVM (in the same EAR)

Stateless Session Bean

Implementation

```
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 implementation (it gives us only literals)

BookFacadeImpl.java

@Stateless : Define Stateless Session Bean EJB type

▷ name : name of the ejb

- used during JNDI lookup
- the value is the *unqualified name* by default (e.g.: BookFacadeImpl)

▷ mappedName : global JNDI name

- vendor specific

▷ description : description

Servlet

Only for testing the local EJB

```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("978-0441717219");
      out.println(book.toString());
    } catch (final FacadeException e) {
      out.println(e.getLocalizedMessage());
    }
    out.close();
  }
}
```

BookPingServlet.java

Inject EJB proxy
@EJB annotation

@EJB : Inject Session Bean proxy

- ▷ Usable only inside the EJB 'context'
  - All EJBs
  - All Servlets (@WebServlet)
  - etc..
- ▷ beanName or lookup : to resolve the EJB implementation
  - Only one of them can be used at the same time
  - If only one implementation exsits, neither attributes should be used (the container resolves the implementation based on the type).
  - beanName : refers the name of the Session bean
  - lookup : refers the JNDI name of the Session bean

The "Modern programming languages" labs of the Óbuda University are necessary and sufficient preconditions of this lab (frontend development).

The lab unofficial site:
http://users.nik.uni-obuda.hu/bedok.david/jse.html

The lab official site:
http://users.nik.uni-obuda.hu/java/

Red Hat® JBoss® Middleware
Enterprise Application Platform

<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 version : **v7.0.0** (2016.05.10.)

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

Commercial product !

Standalone / Domain mode

Contains : <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 version: **v11.0.0.Final** (2017.10.23.)

JEE 7 compliant version: **v10.1.0.Final** (2016.08.19.)

JEE 7 compliant version: **v8.0.0.Final** (2014.02.11.)

Free version of JBoss.

Standalone / Domain mode

JBoss vs. WildFly

The usage of the two application servers are almost the same.

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

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

Start Standalone mode (standalone.xml):

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

Start Standalone mode (use custom.xml):

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

Stop Standalone server instance:

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

Oracle® WebLogic® Server
Enterprise Application Server

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

JEE 7 compliant version: **v12.2.1.3**
JEE 6 compliant version: **v12.1.3**
Commercial product!
It was **BEA**'s product before Oracle acquisition.

It supports the Spring Framework besides JavaEE (a simple *Apache Tomcat®* is enough for the SF in most of the case).

▷ Unzip (`wls1213_dev_update3.zip`)
- e.g.: `c:\work\wls12130\`

▷ Environment variable:
- **MW_HOME** → `c:\work\wls12130\`

▷ Run the installer (*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

Start the domain:

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

Stop the domain:

- ▷ From the WebLogic Server Administration Console or terminate the console window (Ctrl+C)
  - Environment | Servers | Control tab
    - ◦ select "myserver" → Shutdown
      - * When work completes or
      - * Force Shutdown now

ORACLE
WEBLOGIC

▷ WebLogic Server Administration Console
- Deployments | Install..
  ◦ Browse EAR
  ◦ After that you can redeploy the application within a few clicks

The WebLogic creates rich statistics and monitors the running components. It is worth checking the opportunities.

Autodeploy directory:
[MW_HOME]\user_projects\domains\mydomain\autodeploy\

Note: the auto-deployed application cannot be deleted/redeployed via the Deployments page (we have to handle these operations from the autodeploy directory).

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

Copy here: [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

Test
Servlet testing

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

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

The **BookStore** project contains the entire **BookLight** functionality, but for clarity the name of the packages are different.

◈   [gradle|maven]\jboss\bookstore

The root of the **BookStore** can be easily assembled based on the **BookLight**, the development can be continued in the **BookStore** project.
The **BookStore** project will contain the *data tier* (persistence layer).

Root project - Refactor
Redundancy reduction

```
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 [..]
```

The subprojects {...} block is valid for all children build scripts (merge), and there is an allprojects {...} blocks as well, which is valid both the current (it is the *root* now) and the children scripts.

bs-ejbservice subproject:

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

<div align="center">build.gradle</div>

> Because this is the default, this *build* script could be empty.

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 }
```

<div align="center">build.gradle</div>

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

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

For now JBoss 6.4 supports only the Log4j 1.2, WebLogic doesn't support it.

Standard?

Log4J is not part of the JavaEE, the standard uses the **JDK Logging** (of course this is supported by the JBoss and the WebLogic as well).

Simple Logging Facade for Java (**SLF4J**): it can hide the different logging solutions and it is able to handle the configuration dynamically (java.util.logging, logback, log4j).

▷ **Loggers**
  - It connects the Java package, the log level and the appenders (1 logger N appender)
  - So-called `Root Logger` always exists

▷ **Appenders**
  - It defines the pattern of the log message and the type of the logging (e.g.: file, rolling file, e-mail, jms message, etc.)

Configuration

In case of standalone application it can be configured by a **log4j(2).xml** or a **log4j(2).properties** file. In case of Jboss threre is a **Logging subsystem** which wraps it and it can be configured via the **standalone.xml**.

▷ **TRACE**
  - Very detailed data, you can use it sometimes in exceptional cases (e.g.: you need the log messages which were created in the body of a loop).

▷ **DEBUG**
  - In case of debugging or developping. These logs follows through the business workflows in all toched components. With these logs the user activity can be reproduced.

▷ **INFO**
  - These messages are 'always' appear in the log 'file' (we need the information which they deliver). Never use this for frequent events! This is rather a 'Hello, I'am here' type of messages instead of kind of workflow description (you cannot reproduce the user activity from this).

▷ **WARN**
  - Warning. In general case these are kind of bugs which can be handled by the application somehow (e.g.: the transaction performed with certain restrictions).

▷ **ERROR**
  - Fault. It contains as detailed description as it can be. This will be the information when you start to work in a trouble report. Based on this we may ask debug logs to reproduce the user activity.

▷ **FATAL**
  - Catastrophic fault, we use it very rare. The clients don't like to get even Errors, so we don't want to frighten them with Fatals.

```
 1  public class BookPingServlet extends HttpServlet {
 2
 3    private static final Logger LOGGER =
          Logger.getLogger(BookPingServlet.class);
 4
 5    @Override
 6    protected void doGet(HttpServletRequest request,
          HttpServletResponse response) throws ServletException,
          IOException {
 7      LOGGER.info("Get Book by user");
 8      [..]
 9    }
10  }
```

BookPingServlet.java

```java
public class BookFacadeImpl implements BookFacade {

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

  @Override
  public BookStub getBook(String isbn) {
    if (LOGGER.isDebugEnabled()) {
      LOGGER.debug("Get Book (isbn: " + isbn + ")");
    }
    [..]
  }
}
```

alternate solution (less lines of source code):
LOGGER.debug(String.format("Get Book (isbn: %s)", isbn);

BookFacadeImpl.java

Important! In a production environment the DEBUG logs never allow to ruin the performance or the memory, because in general case we don't see that log messages 'production runtime'. The expense of the presented solution is a logical inspection. The alternate solution puts a String literal in the permspace (or metaspace in java 8). We try to avoid the String concatenation (the compiler will replace the concatenation to StringBuilder, but this unnecessary optimalization is very expensive here).

```xml
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

```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

There are separate steps how to setup *Log4J* in WebLogic.
Instead of that we will use **JDK Logging** (part of the JavaEE, java.util.logging.*)

```java
import java.util.logging.Level;
import java.util.logging.Logger;
[..]
private static final Logger LOGGER =
    Logger.getLogger(BookListController.class.getName());
[..]
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	-

If we would like to add an exception beside the error (in case of Log4J there is an overload method):

```
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
Professional open-source database manager

http://www.postgresql.org/

Version : **v10.0** (2017.10.05.)

Version : **v9.6.5** (2017.09.31.)

Install : installer

Default data : **5432** (port) és **postgres** (database)

Default user : **postgres** (user) és **root** (password)

Editor : **pgAdmin III** vagy **pgAdmin IV**

Environment variables :

- ▷ **PSQL_HOME** →c:\ProgramFiles\PostgreSQL\9.4
- ▷ Modify **Path** → %Path%;%PSQL_HOME%\bin

Professional tool

Besides that the PostgreSQL is as professional tool like the Oracle DB or the SAP Adaptive Server Enterprise (earlier Sybase ASE), there are active PostgreSQL systems in production environments that manage in excess of 4 terabytes of data.

PostgreSQL - psql
Connect

Version information

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

Connect (CLI)

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

Connect and execute commands from file

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

Oracle MySQL
The most popular open-source database manager

MySQL

https://www.mysql.com/
Download: https://dev.mysql.com/downloads/
The Community version is free (GPL).
Version: **v5.7.20**
Default data: **3306** (port)
Default user: **root** (user) és - (password)
Editor: **MySQL Workbench**
Encironment variables:

  ▷ **MYSQL_HOME** →c:\ProgramFiles\MySQL\5.7\bin
  ▷ Modify **Path** → %Path%;%MYSQL_HOME%\bin

Popular tool

In the world of PHP development the MySQL is almost essential (almost all the Shared Hosting solution support the MySQl as persistence layer). MySQL contains several engines, all of them have advantages/disadvantages. Perhaps InnoDB could be a professional engine (but it has some know issues, and it has not supported *role*s yet). *role*-okat ez sem kezel még.

BookStore db scripts
Create Database, Role and User

◈   [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

Execute BookStore db scripts
Create Database, Role and User

```
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
```

Connect the to **bookstoredb** database with the **bookstore_user** user

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

BookStore db schema

Book and Book's category

```sql
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: there will be a
> book_book_id_seq   sequence
> in the database.

create-schema.sql

```
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 ;
```

<div align="center">grant-access.sql</div>

```
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 - Data loading
Create initial data

```
1 INSERT INTO bookcategory (bookcategory_id, bookcategory_name)
      VALUES (0, 'SCIFI');
2 INSERT INTO bookcategory (bookcategory_id, bookcategory_name)
      VALUES (1, 'LITERATURE');
3 INSERT INTO bookcategory (bookcategory_id, bookcategory_name)
      VALUES (2, 'HISTORICAL');
4 INSERT INTO bookcategory (bookcategory_id, bookcategory_name)
      VALUES (3, 'PHILOSOPHY');
5
6 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);
7 [..]
```

initial-data.sql

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

Access database in Java

JDBC

This is a Java SE topic in practice, we have to use the `java.sql` package to reach that goal. Besides we need the **JDBC driver** of the choosen database engine (**Java DataBase Connectivity**, typically it is a `jar` file) (this is the Java version of the ODBC (Open Database Connectivity)).

```java
Class.forName("org.postgresql.Driver");
Connection connection =
    DriverManager.getConnection("jdbc:postgresql://localhost:5432/bo
    "bookstore_user", "123topSECret321");

Statement statement = connection.createStatement();
ResultSet rs = statement.executeQuery("SELECT book_id, book_title
    FROM book");
while (rs.next()) {
  long id = rs.getLong("book_id");
  String title = rs.getString("book_title");
}
rs.close();
statement.close();

connection.close();
```

This sample code completely ignores the error handling. Real example can be found in the `shopping` project. The greatest disadvantage of the native JDBC driver usage is the **non type-safe** behavior.

Java Persistence API
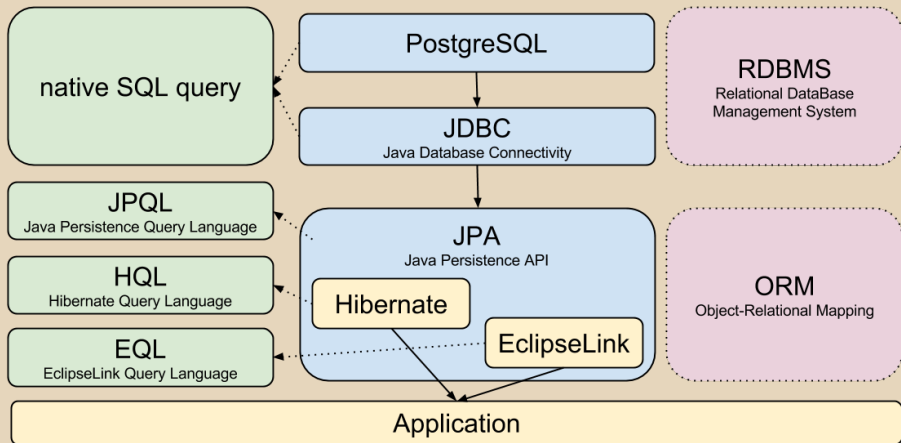
▷ Create **new module**: **PostgreSQL JDBC driver**

▷ **Regsiter module** for the `standalone domain`

▷ **Register database driver** inside the `standalone domain`

▷ **Create datasource** for the **bookstoredb** inside the `standalone domain`

JBoss - Create new module
For the JDBC driver of PostgreSQL

⬥ environment\modules\org\postgresql\main\

Directory structure

```
[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

```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 - Register database driver
Standalone domain

```xml
<subsystem xmlns="urn:jboss:domain:datasources:1.2">
  <datasources>
    [..]
    <drivers>
      [..]
      <driver name="postgresql" module="org.postgresql">
        <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-da
      </driver>
    </drivers>
  </datasources>
</subsystem>
```

standalone.xml

JBoss - Create datasource
For the PostgreSQL `bookstoredb` database

[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 - Connection validation after DB restart
Modify the configuration of the BookStore datasource

[JBOSS_HOME] | **standalone** | configuration | standalone.xml

```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

▷ Add PostgreSQL JDBC library into the domain's classpath

- Copy JDBC driver (e.g.: `postgresql-9.4-1201.jdbc41.jar`) here: [MW_HOME]\user_projects\domains\mydomain\lib\
- Restart server instance

▷ 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**
  - It is worth testing the connection through the administration console

Persistence
New subproject

Directory structure

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

```
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' }
```

<div align="center">build.gradle</div>

JPA implementation

The JBoss 6.4 uses the **Hibernate 4.3.10.Final** at runtime as a JPA implementation, but we would like to use only the JPA API during development, so we don't need this dependency:
'org.hibernate:hibernate-core:4.3.10.Final'

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

<div align="center">build.gradle</div>

Dependencies

bs-weblayer uses bs-ejbservice to reach the BookFacade EJB.
bs-ejbservice uses bs-persistence to reach the BookService EJB.

hibernate.dialect: not mandatoy, but recommended (is we set it the Hibernate is able to generate PostgreSQL specific native queries too. If we omit in this example, everything will remain fine (ANSI SQL is enough at this time).

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

```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-sou
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>
```

Name of the **persistence unit**: bs-persistence-unit
JNDI name: java:jboss/datasources/bookstoreds

ORACLE
WEBLOGIC

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

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

The Oracle WebLogic uses **EclipseLink** (TopLink) as JPA Provider. In the world of JavaEE we can freely move among servers, our 'only' task is change/fine-tune the descriptors.

persistence.xml

Book entity
Identification field

If we do not set the `@SequenceGenerator` and the `@GeneratedValue` annotations, we will get an `org.hibernate.id.IdentifierGenerationException` during insert: "ids for this class must be manually assigned before calling save()".

```
1  package hu.qwaevisz.persistence.entity;
2                   Database table name: book
3  @Entity
4  @Table(name = "book")
5  public class Book implements Serializable {
6                   Database sequence name: book_book_id_seq
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                  Database column name: book_id
13   [..]
14 }
```

Book.java

Book entity
Further fields

```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;
```

In the level of the database the **bookcategory** is a table, but in ORM it will be an enum (CRUD operations are not allowed on it).

```java
@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;
```

Book.java

BookCategory
enum in the level of ORM

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

public enum BookCategory {

  SCIFI,
  LITERATURE,
  HISTORICAL,
  PHILOSOPHY;

}
```

BookCategory.java

▷ The **BookStore** application defines services in two different levels.

- The EJB services of the bs-ejbservice project's level support the 'customer' needs.
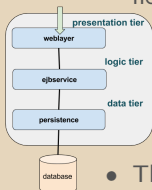
  ◦ The methods of the interface serve business *use-case*s.
  ◦ The types used by the interface are **stub**s. These are open to the public in technical level.
  ◦ The exceptions used by the interface are open to the public or these can be converted into public error messages. Its rarely contain technical details (security).

- The EJB services of the bs-persistence project's level serve the 'developers' needs (primarily formed by technical requirements and decisions).

  ◦ The methods of the interface serve the developers who try to keep the balance among the redundancy free, maintainable, clean and sometimes future-proof requirements.
  ◦ The types used by the interface are ofter **entities** or internal DTOs. These are not open to the public.
  ◦ The exceptions used by the interface can contain detailed technical information. Most of the time these will not appear in the 'client' side.

```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
Implementation

```
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")
Name of the parameter: isbn
getSingleResult(): exactly one record exists (otherwise it will be thrown an exception)
getResultList(): List<Book> will be the result value

JPA Named Query
JPQL syntax

```java
@Entity
@Table(name = "book")
@NamedQueries(value = { //
    @NamedQuery(name = Book.GET_BY_ISBN, query = "SELECT b FROM
        Book b WHERE b.isbn=:isbn"),
    @NamedQuery(name = Book.GET_ALL, query = "SELECT b FROM Book
        b ORDER BY b.title"),
    [..]
})
public class Book implements Serializable {

  public static final String GET_BY_ISBN = "Book.getByIsbn";
  public static final String GET_ALL = "Book.getAll";

  [..]
}
```

Book.java
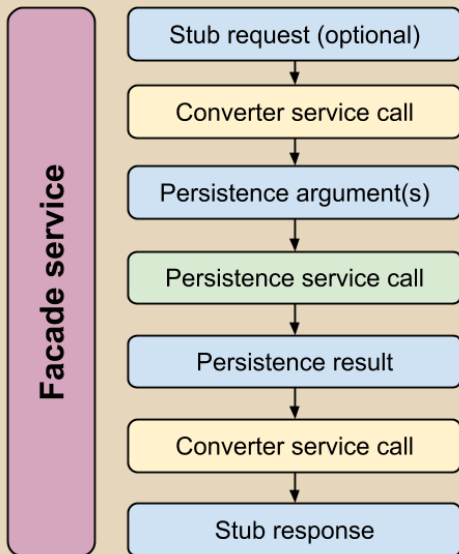
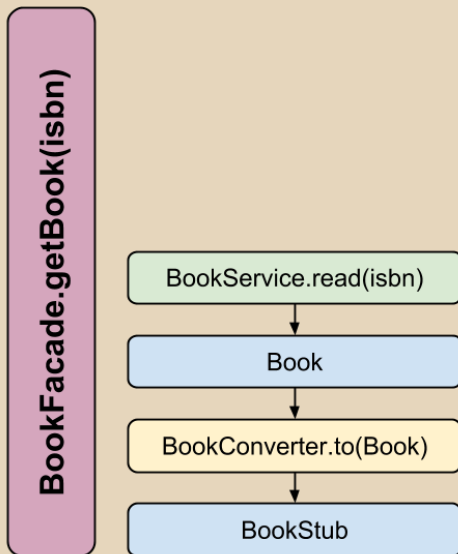# BookConverter

EJB Service project (`bs-ejbservice`)

```
1  @Local
2  public interface BookConverter {
3    BookStub to(Book book);
4    List<BookStub> to(List<Book> books);
5  }
```

BookConverter.java

Very good 3**rd** party libraries exists for conversions, e.g.: **MapStruct** (http://mapstruct.org/).

```
1  @Stateless
2  public class BookConverterImpl implements BookConverter {
3
4    @Override
5    public BookStub to(Book book) {
6      final BookCategoryStub category =
             BookCategoryStub.valueOf(book.getCategory().toString());
7      return new BookStub(book.getIsbn(), book.getAuthor(), book.getTitle(), category,
             book.getPrice(), book.getNumberOfPages());
8    }
9    [..]
10 }
```

BookConverterImpl.java

BookFacade modification

EJB Service project (`bs-ejbservice`)

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

  @EJB
  private BookService service;

  @EJB
  private BookConverter converter;

  @Override
  public BookStub getBook(String isbn) throws FacadeException {
    try {
      final BookStub stub = this.converter.to(this.service.read(isbn));
      if (LOGGER.isDebugEnabled()) {
        LOGGER.debug("Get Book by isbn (" + isbn + ") --> " + stub);
      }
      return stub;
    } catch (final PersistenceServiceException e) {
      LOGGER.error(e, e);
      throw new FacadeException(e.getLocalizedMessage());
    }
  }
  [..]
}
```

BookFacadeImpl.java

Earlier we had only one test servlet (BookPingServlet) to call a local EJB
service. Create the followings based on the *Modell-View-Controller* design.

▷ **BookController** servlet which get the ISBN number as an URI parame-
ter then fetches the BookStub instance and puts it in the HttpServletReq
and *forward*s the request to a *JSP*.

▷ **book.jsp** which gets the BookStub instance from the HttpServletReques
and builds an XHTML output.

BookController

Weblayer project (`bs-weblayer`)

```java
package hu.qwaevisz.bookstore.weblayer.servlet;
[..]
@WebServlet("/Book")
public class BookController extends HttpServlet {
  [..]
  @EJB
  private BookFacade facade;

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
    String isbn = request.getParameter("isbn");
    [..]
    try {
      BookStub book = this.facade.getBook(isbn);
      request.setAttribute("book", book);
    } catch (final FacadeException e) {
      LOGGER.error(e, e);
    }
    RequestDispatcher view = request.getRequestDispatcher("book.jsp");
    view.forward(request, response);
  }
  [..]
}
```

BookController.java

```
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

Further tasks:

- ▷ List all of the books(JPA query, persistence service, facade service, converter ejb)
  - Filter the books via Criteria (optional)
- ▷ `list.jsp` and `BookListController` servlet realization
- ▷ Bind `list.jsp` and `book.jsp` (link the controller servlets)
- ▷ Create a `page.css` file for the webpages
- ▷ ...

The **BookStore** application contains the full of CRUD operations in addition to queries. The technical details of this will be coverd later.

The next few slides presents the affected persistence operations. The solutions lack transaction- and error handling. Later we will cover this during the **school** project.

BookStore - Record a new book

BookStore webapplication

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

  @Override
  public Book create(final String isbn, final String author, final String title, final
      int numberOfPages, final double price, final BookCategory category)
    throws PersistenceServiceException {
    [..]
    final Book book = new Book(isbn, author, title, numberOfPages, price, category);
    this.entityManager.persist(book);
    [..]
    return book;
  }
  [..]
}
```

BookServiceImpl.java

BookStore - Modify a book
BookStore webapplication

```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

BookStore - Remove a book

BookStore webapplication

```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

The referred **JPQL**:

DELETE FROM Book b WHERE b.isbn=:isbn

Testing
Level of testing

The unit-testing is always part of the coding development process. This lab does not cover the unit-testing during implementation, because this is outside the main scope.

- ▷ **unit** or component test: in an object-oriented application one class (one responsibility) is an independent part of the system and we have to verify its behavior without any internal and external dependencies. The main goal of this testing is to find program faults quickly, and supports the refactoring (if the *coverage* is enough high).

- ▷ **integration** test: if we test some components together it could accelerate the development. We have to decide which dependencies of the environment is mocked/skipped and which does not.

- ▷ **system** test: tipically the highest level of testing which are created by the developers. The system or a standalone part of the system will be tested in a close to the real environment (there will be simulators also).

- ▷ **end-to-end** test: in a well-functioning team/company this level of testing are handled by a separate test team (integration and verification team). If there are engineers/developers inside that team, automated test cases will be created (e.g.: *Selenium* tests in case of a webapplication), but the manual test cases are common too.

Testing the persistence layer

▷ It is cumbersome enough to deploy the application just for testing that a named query is fine or not (but: easy to make a mistake during named query creation).

▷ Database operations and `named query` testing is not a unit-test responsibility.

▷ JPA API can be used in a *standalone* Java application too. This could be a way to test the named queries, but it could be inconvenient also and this is not an automatic solution.

▷ Propable the most elegant way to create **integration test**s for the persistence layer. The written named queries and entity manager operations are really executed on the database, the efficiency and performance could be measured and checked, but we don't need to use the application server at all.

▷ In general the integration and the unit test do not run at the same time. We have to find a way how to separate these tests (there are many different solutions).

▷ The classes of the persistence layer (where it is meaningful and not boilerplate algorithms are affected) have to cover by unit-tests of course. We will learn about EJB services' unit-testing later in the **school** project.

Standalone Persistence Unit for the integration tests

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        <property name="jboss.as.jpa.managed" value="false" />
17      </properties>
18    </persistence-unit>
19 </persistence>
```

The JPA reads the persistence.xml from a fix path of the classpath. That's way we add the new persistent unit in the same configuration file, and that's way we have to use the jboss.as.jpa.managed value (false). JBoss will ignore that standalone persistence unit (it causes ClassNotFound error). We have to set the **provider** class and the connection details instead of the JNDI name. And finally we have to list the entities because there is no container who looks through the classes of the classpath (to find @Entity annotations).
Name of the **persistence unit**: bs-persistence-test-unit

Classpath of the integration tests

The integration tests run without the JavaEE container (e.g.: gradle/maven/eclipse's JVM runs those), so we need not only the interfaces/enums of the JavaEE. Beneficial using the JavaEE dependency of the target environment, but we can choose anything else too.

Required runtime dependencies:

- ▷ **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 implementation**
    - `org.jboss.spec:jboss-javaee-6.0:3.0.3.Final`
    - **Important!** We cannot add the JavaEE 6.0 API into the classpath at the same time:
        - ◦ `javax:javaee-api:6.0`

```
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 [..]
```

> configure(..): It is valid for all subprojects except the bs-persistence, where the Java EE 6.0 implementation will be add to the testCompile's classpath.

build.gradle

```
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

The **compileOnly** dependency configuration has been part of the *java plugin* since Gradle 2.12. So the JavaEE API will not be part of the testCompile's classpath (it would spoil the tests). Good - but not as elegant - solution putting the Java EE 6.0 implementation into the compile's classpath.

src | **test** | **resources** | log4j.xml

```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

The detailed logs are very useful during implementation and testing. During the integration test running there is no JavaEE container which initialize the logging, we have to do that instead.

Integration test
Preparations

```
 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  }
```

We have to create a default/protected setEntitymanager() mutator method in the production source to "inject" the local entityManager :(

BookServiceImplIntegrationTest.java

Integration test

Test the query behind the `read()` method

> We have to differentiate the integration test methods with the `integration` group.

```java
public class BookServiceImplIntegrationTest {
  [..]
  @Test(groups = "integration")
  private void readSampleBookFromDatabase() throws PersistenceServiceException {
    final Book book = this.object.read("978-0441172719");
    this.assertBook(book, "978-0441172719", "Frank Herbert", "Dune",
        BookCategory.SCIFI, 896, 3500d);
  }

  private void assertBook(final Book book, final String isbn, final String author,
      final String title, final BookCategory category,
      final Integer numberOfPages, final double price) {
    Assert.assertEquals(book.getIsbn(), isbn);
    Assert.assertEquals(book.getAuthor(), author);
    Assert.assertEquals(book.getTitle(), title);
    Assert.assertEquals(book.getCategory(), category);
    Assert.assertEquals(book.getNumberOfPages(), numberOfPages);
    Assert.assertEquals(book.getPrice(), price);
  }
  [..]
}
```

BookServiceImplIntegrationTest.java

Integration test
Data manipulation in transaction

```
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 }
```

There is no EJB container which can handle the transactions.
We have to manage that in case of data manipulation. It would
be good if the test will be *rerunable* as well.

BookServiceImplIntegrationTest.java

```
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

There are several ways to run integration tests :
- ▷ Create a separate **integration project** (perhaps the most elegant solution)
- ▷ Create a separate configuration **profil** (separate classpath, testng config, etc.)
- ▷ Define separate TestNG `suite.xml` files for the unit- and the integration tests (for simplicity we will choose that)
- ▷ Exclude/Include globally some TestNG's group to select test cases

Integration TestNG suites

src | test | **resources** | testng-*.xml

```
1  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
2  <suite name="integration-test-suite" verbose="1" >
3    <test name="bookservice">
4      <classes>
5        <class name="hu.[..].service.BookServiceImplIntegrationTest" />
6        <class name="hu.[..].service.BookSearchImplIntegrationTest" />
7      </classes>
8    </test>
9  </suite>
```

testng-integration.xml

```
1  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
2  <suite name="unit-test-suite" verbose="1">
3    <test name="bookservice">
4      <groups>
5        <run>
6          <exclude name="integration" />
7        </run>
8      </groups>
9      <packages>
10       <package name="hu.[..].service.*" />
11     </packages>
12   </test>
13 </suite>
```

All the methods which located in the named package are unit tests, except which belongs to the integration group.

testng-unit.xml