UNIVERSITÀ DI PISA

AI-DII

Hands-on session 5: Glassfish & EJBs

Distributed Systems and Middleware Technologies

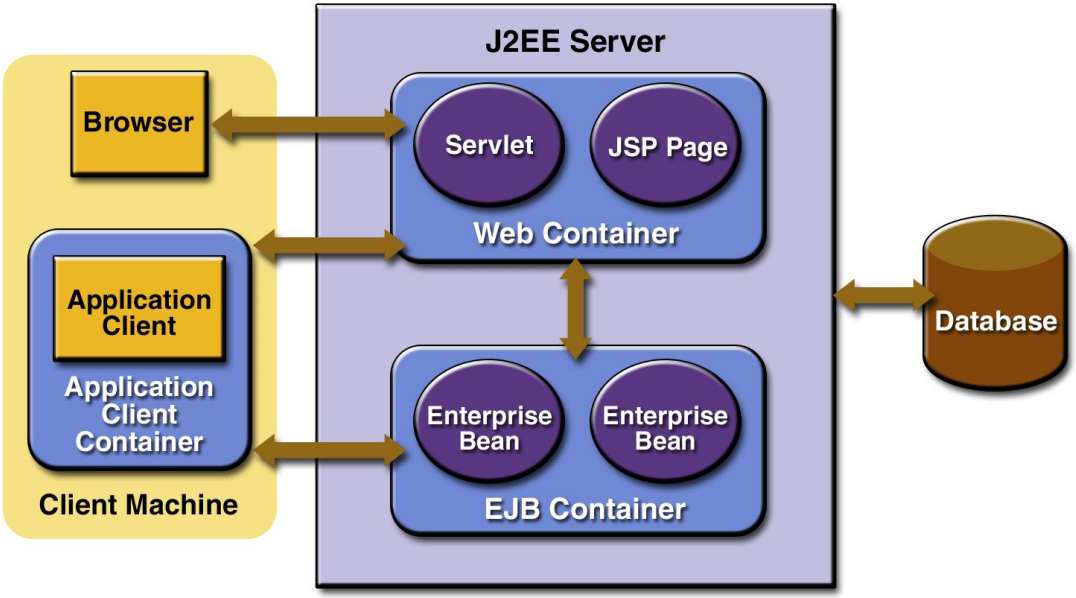
2023/2024

By José Luis Corcuera Bárcena

Agenda

1. Java Enterprise Servers
2. Glassfish Application Server
3. Configuring IntelliJ to work with Glassfish
4. Deploying our Web Application in Glassfish
5. Step 1: creating all projects
6. Step 2: implementing the interfaces - pojos module
7. Step 3: implementing the EJB Application module
8. Step 4: implementing the Web Application module
9. Step 5: deploying our EJB & Web Application in Glassfish
10. Exercises

Java Enterprise Servers



Apache Tomcat = Web Container.

Glassfish = Web Container + EJB Container.

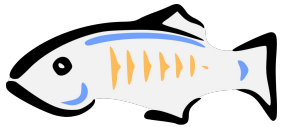
Image taken from: https://serverguy.com/servers/open-source-java-ee-application-servers/

Glassfish Application Server (1)

- GlassFish is an open-source Jakarta EE platform application server project.
- Built by Sun Microsystems, then sponsored by Oracle Corporation, and now living at the Eclipse Foundation and supported by Payara, Oracle and Red Hat.
- Project home page: https://glassfish.org/

For our hands-on session, we are going to use Glassfish 7.0.10

Glassfish Application Server (2) - directory structure



```
jose@uss-defiant:~/software/glassfish-7.0.10$ tree -L 1
.
├── bin
├── glassfish
├── javadb
├── META-INF
├── mq
└── README.txt

5 directories, 1 file
jose@uss-defiant:~/software/glassfish-7.0.10$ pwd
/home/jose/software/glassfish-7.0.10
```

bin: executables files.

glassfish: Server directory. It defines main files of this server (dependencies, domains, configuration files, etc.).

javadb: Java database.

META-INF: Metadata information.

mq: Message Queue home directory.

Glassfish Application Server (3) - running the server

- Open a terminal and move to your **${GLASSFISH_HOME}/bin** directory.
- Grant executable permissions to all **executable** files by execution the following command:

chmod +x *

- To start the server run:

./asadmin start-domain

- To stop the server run:

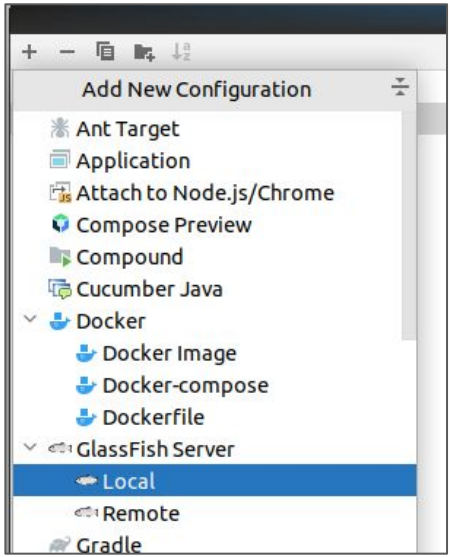
./asadmin stop-domain

- To watch the log files of the default domain run the following command:

tail -f ${GLASSFISH_HOME}/glassfish/domains/domain1/logs/server.log

Configuring IntelliJ to work with Glassfish

We can deploy our Web Application directly in Glassfish by creating a new Configuration descriptor.

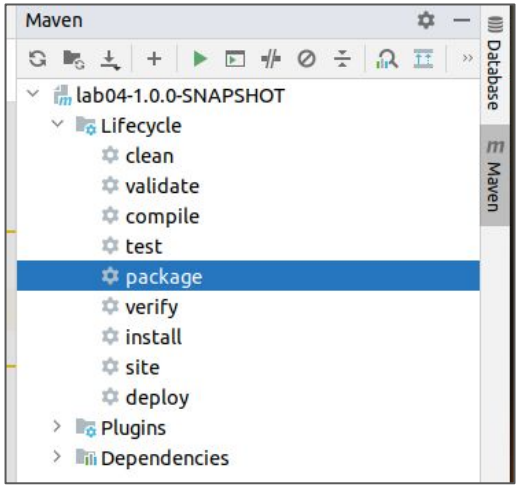


A new window will be shown and you can finish to configure the deployment of your application like we did in the previous hands-on session (same steps).

Add New Configuration -> GlassFish Server -> Local

Deploying our Web Application in Glassfish (1)

- Run the **package** maven lifecycle phase on your project.

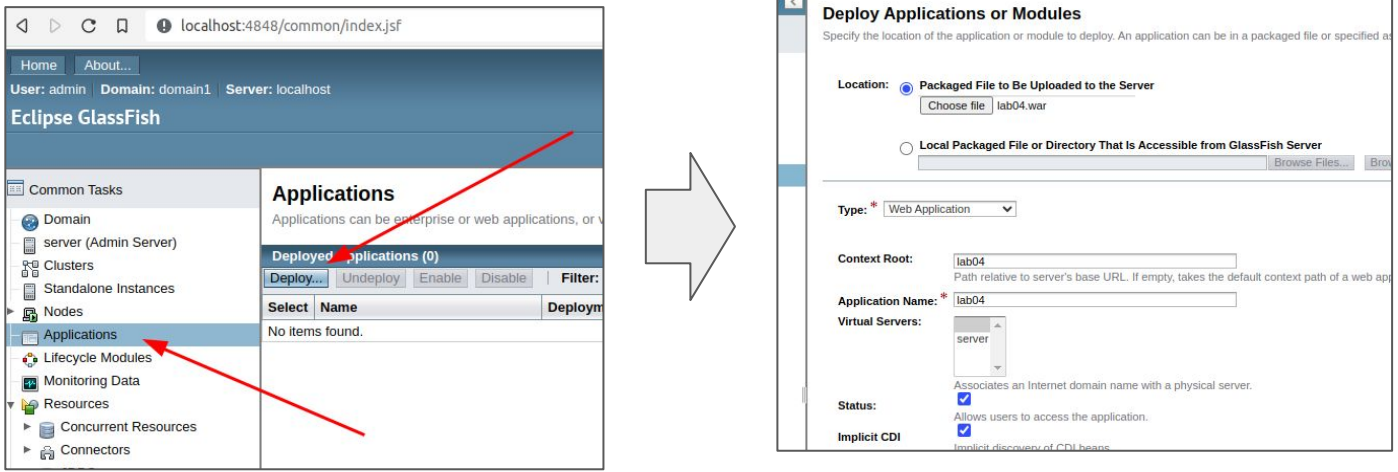


- As a result, a **war** file will be generated.

```
[INFO] Copying webapp resources [/home/jose/repository/UNIPI/DSMT/unipi-dsmt-2023-2024/lab04/src/main/webapp]
[INFO] Building war: /home/jose/repository/UNIPI/DSMT/unipi-dsmt-2023-2024/lab04/target/lab04.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

Deploying our Web Application in Glassfish (2)

- Access to the Glassfish Administration console (http://localhost:4848/) and proceed with the deployment by uploading the generated **war** file.



The credentials to access to this Administration console are **username = "admin"** and **leave the password blank**.

Exercise 01: Running Glassfish & Deploying a Web Application - 10 min

In this exercise it is required to **deploy** our lab04 web application into a Glassfish installation. **You must perform the deployment directly in Glassfish by using the Administration console**.

Exercise 02: Deploying a Web Application in Glassfish by using IntelliJ

Similar to the previous exercise, you have to configured your IntelliJ IDE to deploy lab04 web application in Glassfish. **Be careful, remember to stop the Glassfish instance in the previous exercise**.

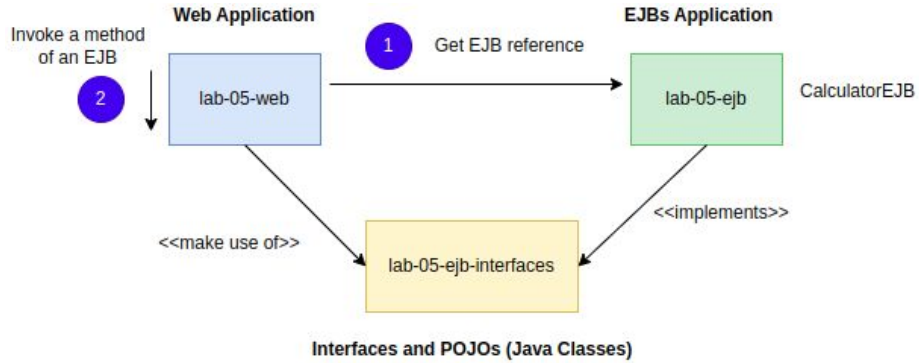
Java Enterprise Application Components

Projects description:

lab-05-web: It defines JSP, static resources, Servlets, etc.

lab-05-ejb: It defines EJB (Our Business Logic).

lab-05-ejb-interfaces: It defines the method signatures and data types to be used in the EJBs definition.

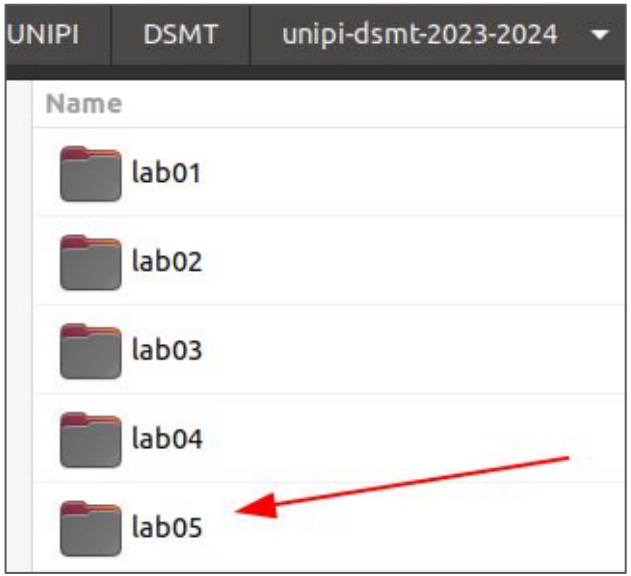


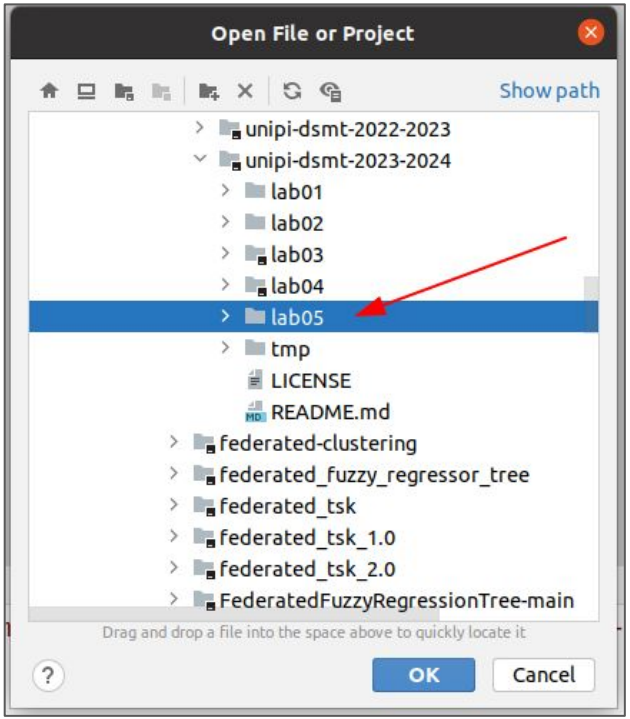

```java
import jakarta.ejb.Remote;
@Remote
public interface CalculatorEJB {
    int sum(int a, int b);
    int sub(int a, int b);
    int mul(int a, int b);
    double div(int a, int b);
}
```

Step 1: creating all projects (1)

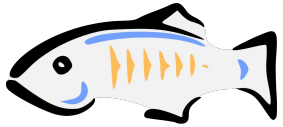
As first step, let's define a folder that will contain our projects and **open it in IntelliJ**

Folder: lab05



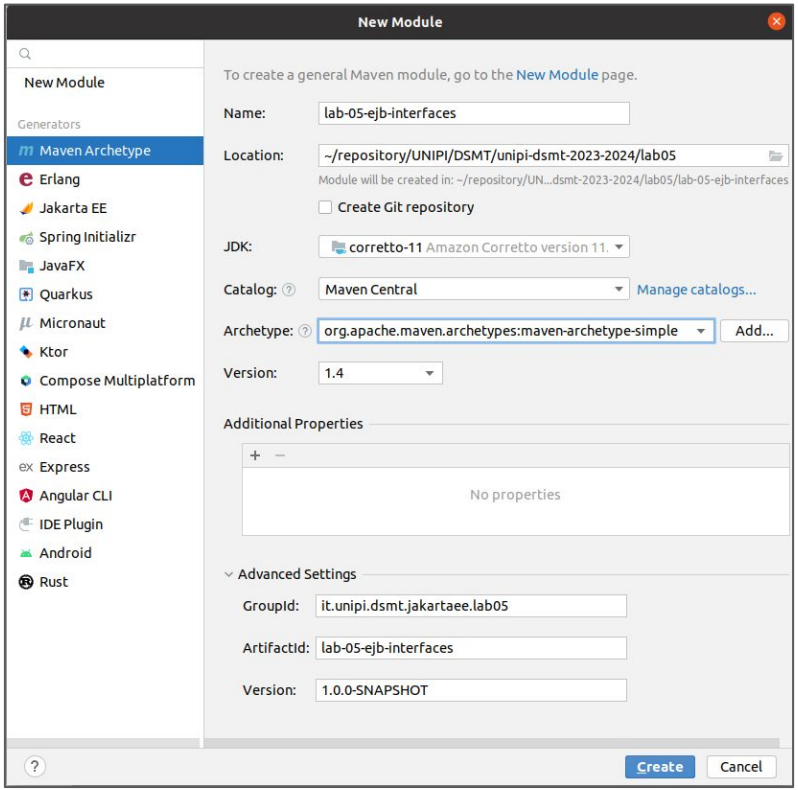


Step 1: creating all projects (2)



Next, we have to create a new module for the interfaces and pojos (Java classes) to interact with the EBJs.

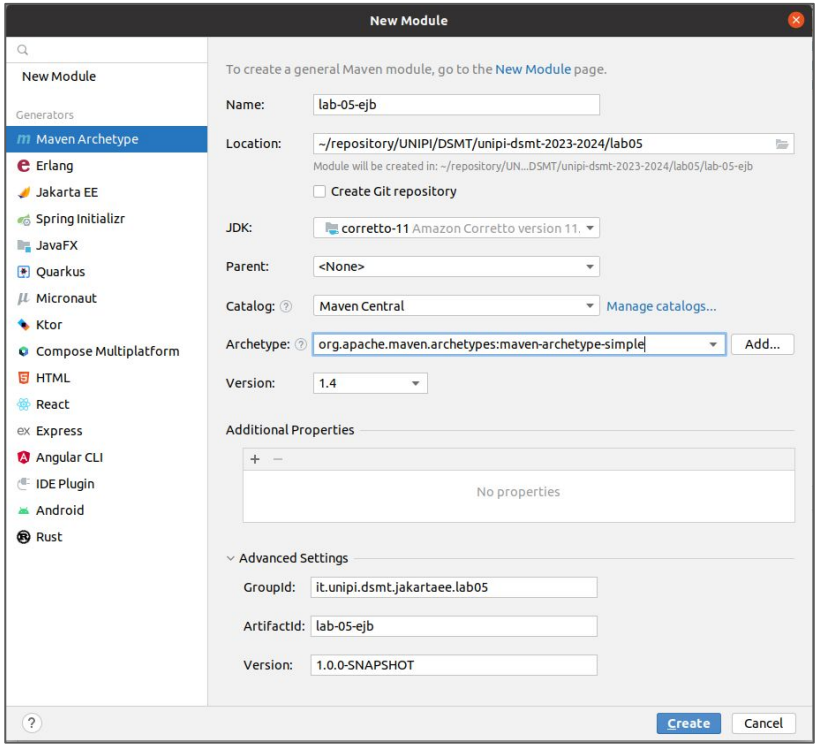
Project: lab-05-ejb-interfaces



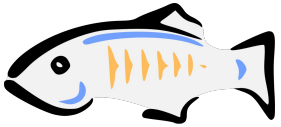
Step 1: creating all projects (3)

Later, we have to define a new module which implements the interfaces defined in the previous module (same steps).

Project: lab-05-ejb

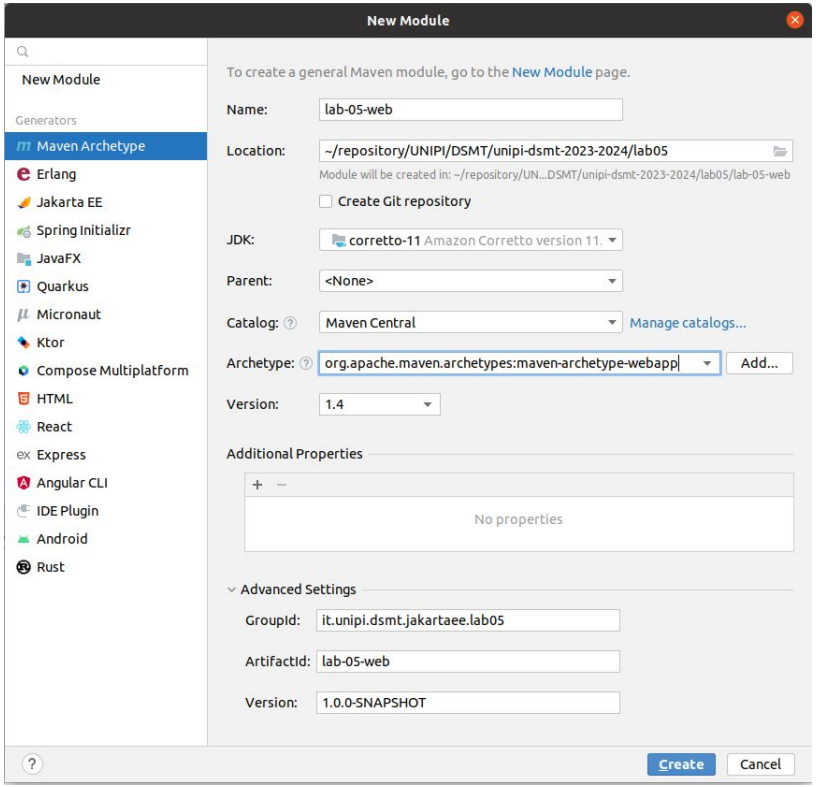


Step 1: creating all projects (4)



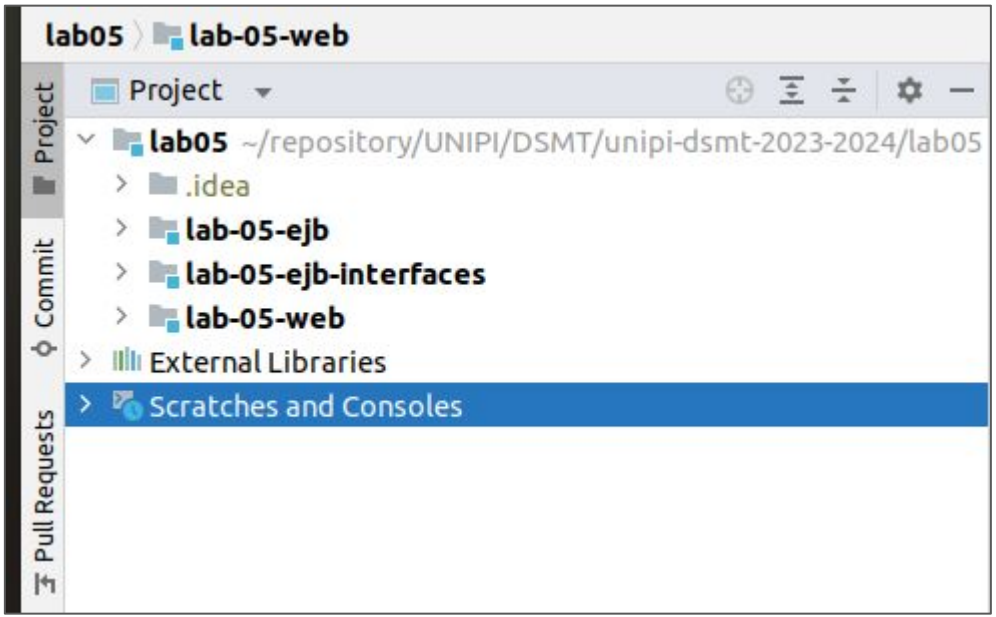
Finally, in this step, we create a web application module

Project: lab-05-web

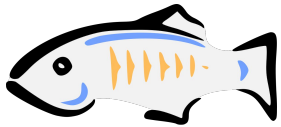


Step 1: creating projects (5)

Expected project structure



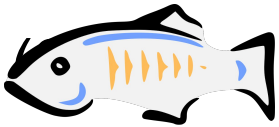
Exercise 03: JakartaEE project creation



In this exercise you have to create a folder which contains the 3 modules described in the previous slides.

5 minutes

Step 2: implementing the interfaces - pojos module



After every code update in this module, run the "install" lifecycle.
Important: If you define POJOs, make sure they implement the interface Serializable.

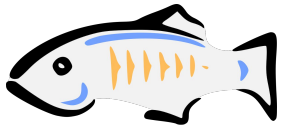
```
<groupId>it.unipi.dsmt.jakartaee.lab05</groupId>
<artifactId>lab-05-ejb-interfaces</artifactId>
<version>1.0.0-SNAPSHOT</version>

<name>${artifactId}-${version}</name>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEn
coding>
 <maven.compiler.source>11</maven.compiler.source>
 <maven.compiler.target>11</maven.compiler.target>
</properties>

<dependencies>
 <dependency>
 <groupId>jakarta.platform</groupId>
 <artifactId>jakarta.jakartaee-api</artifactId>
 <version>10.0.0</version>
 <scope>provided</scope>
</dependency>

</dependencies>
```

Piece of code
pom.xml

```
package it.unipi.dsmt.jakartaee.lab05.interfaces ;

import jakarta.ejb.Remote;

@Remote
public interface CalculatorEJB {
    int sum(int a, int b);
    int sub(int a, int b);
    int mul(int a, int b);
    double div(int a, int b);
}
```

CalculatorEJB.java

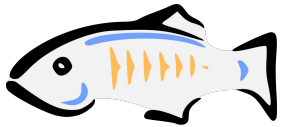
Exercise 04: interfaces and POJOs module implementation



In this exercise you have to implement the CalculatorEJB in the lab-05-ejb-interfaces module. You can find the pom.xml file and the java file in the folder lab05/files/lab-05-ejb-interfaces.

5 minutes

Step 3: implementing the EJB Application module



Select the proper EJB annotation for your EJBs.

```xml
<groupId>it.unipi.dsmt.jakartaee.lab05</groupId>
<artifactId>lab-05-ejb</artifactId>
<version>1.0.0-SNAPSHOT</version>

<name>${artifactId}-${version}</name>

<dependencies>
 <dependency>
   <groupId>it.unipi.dsmt.jakartaee.lab05</groupId>
   <artifactId>lab-05-ejb-interfaces</artifactId>
   <version>1.0.0-SNAPSHOT</version>
 </dependency>
<dependency>
 <groupId>jakarta.platform</groupId>
 <artifactId>jakarta.jakartaee-api</artifactId>
 <version>10.0.0</version>
 <scope>provided</scope>
</dependency>

</dependencies>
```

Piece of code
pom.xml

```java
package it.unipi.dsmt.jakartaee.lab05.ejb;

import it.unipi.dsmt.jakartaee.lab05.interfaces.CalculatorEJB;
import jakarta.ejb.Stateless;

@Stateless
public class CalculatorEJBImpl implements CalculatorEJB {
    @Override
    public int sum(int a, int b) {
        return a + b;
    }
    @Override
    public int sub(int a, int b) {
        return a - b;
    }
    @Override
    public int mul(int a, int b) {
        return a * b;
    }
    @Override
    public double div(int a, int b) {
        return a * 1.0 / b;
    }
}
```

CalculatorEJBImpl.java

How can this project have visibility/access to this CalculatorEJB?

Exercise 05: EJB module implementation

In this exercise you have to implement the CalculatorEJBImpl in the lab-05-ejb module. You can find the pom.xml file and the java file in the folder lab05/files/lab-05-ejb.

5 minutes

Step 4: implementing the Web Application module (1)

In our Web Application module, EJBs references are obtained by using the **@EJB annotation**. **Module: lab-05-web**

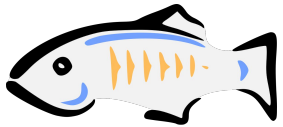
```
<groupId>it.unipi.dsmt.jakartaee.lab05</groupId>
<artifactId>lab-05-ejb</artifactId>
<version>1.0.0-SNAPSHOT</version>

<name>${artifactId}-${version}</name>

<dependencies>
 <dependency>
   <groupId>it.unipi.dsmt.jakartaee.lab05</groupId>
   <artifactId>lab-05-ejb-interfaces</artifactId>
   <version>1.0.0-SNAPSHOT</version>
 </dependency>
<dependency>
 <groupId>jakarta.platform</groupId>
 <artifactId>jakarta.jakartaee-api</artifactId>
 <version>10.0.0</version>
 <scope>provided</scope>
</dependency>
</dependencies>
```

Piece of code
pom.xml

```
import jakarta.ejb.EJB;
import jakarta.servlet.ServletException ;
import jakarta.servlet.annotation. WebServlet;
import jakarta.servlet.http.HttpServlet ;
import jakarta.servlet.http.HttpServletRequest ;
import jakarta.servlet.http.HttpServletResponse ;

import java.io.IOException ;

@WebServlet(name = "CalculatorServlet" , value = "/CalculatorServlet" )
public class CalculatorServlet extends HttpServlet {

   @EJB
   private CalculatorEJB calculatorEJB ;

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException , IOException {
       int a = Integer.parseInt(request.getParameter( "a"));
       int b = Integer.parseInt(request.getParameter( "b"));
       String action = request.getParameter( "action");
       double result = 0;
       switch(action){
           case "add":
               result = calculatorEJB.sum(a, b);
```

Piece of code
CalculatorServlet.java

Step 4: implementing the Web Application module (2)



In our Web Application, EJBs references are obtained by using **JNDI**. **Project: lab-05-web**

Portable JNDI names for EJB CalculatorEJBImpl:
[**java:global/lab_05_ejb/CalculatorEJBImpl!it.unipi.dsmt.jakartaee.lab05.interfaces.CalculatorEJB**,
java:global/lab_05_ejb/CalculatorEJBImpl]]]

Glassfish-specific (Non-portable) JNDI names for EJB CalculatorEJBImpl:
[it.unipi.dsmt.jakartaee.lab05.interfaces.CalculatorEJB#it.unipi.dsmt.jakartaee.lab05.interfaces.CalculatorEJB,
it.unipi.dsmt.jakartaee.lab05.interfaces.CalculatorEJB]]]

${GLASSFISH_HOME}/glassfish/domains/domain1/logs/server.log

```java
@WebServlet(name = "CalculatorServlet", value = "/CalculatorServletJNDI")
public class CalculatorServletJNDI extends HttpServlet {

    public CalculatorEJB getCalculatorEJB() {
        Properties props = new Properties();
        InitialContext ic = null;
        try {
            ic = new InitialContext(props);
            String id = "java:global/lab_05_ejb/CalculatorEJBImpl!" + CalculatorEJB.class.getName();
            System.out.println("Id: " + id);
            return (CalculatorEJB) ic.lookup(id);
        } catch (NamingException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
        CalculatorEJB calculatorEJB = getCalculatorEJB();
        int a = Integer.parseInt(request.getParameter("a"));
        int b = Integer.parseInt(request.getParameter("b"));
        String action = request.getParameter("action");
        double result = 0;
        switch(action){
            case "add":
                result = calculatorEJB.sum(a, b);
```

FortuneCookieServlet.java

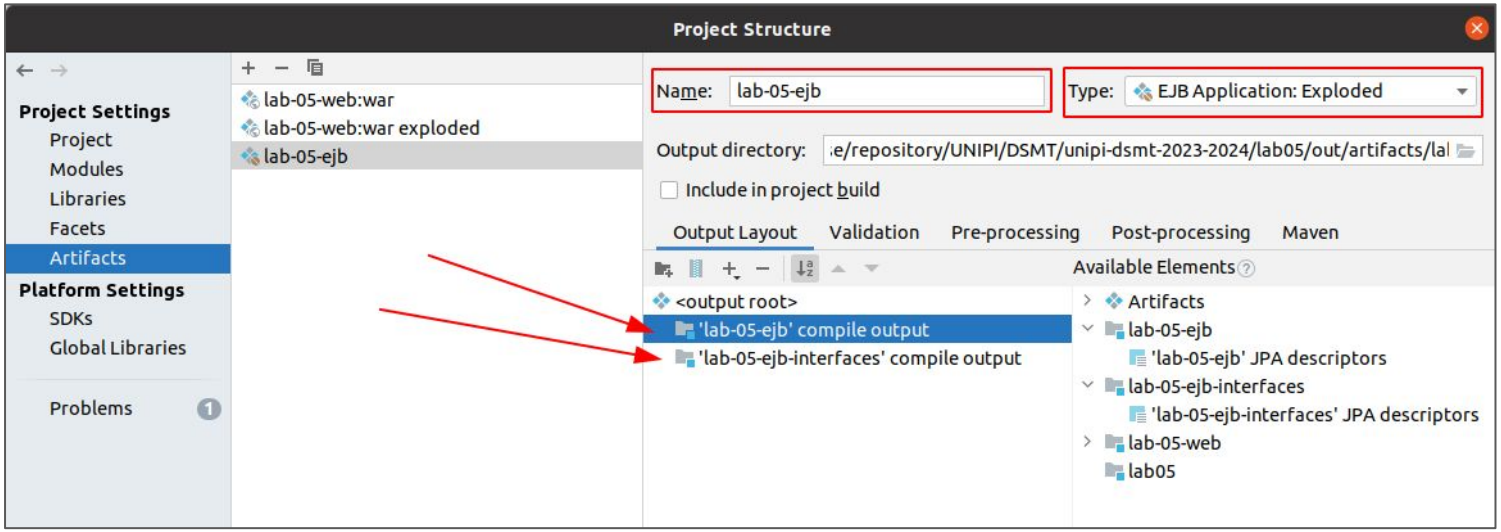
Exercise 06: web application module implementation

In this exercise you have to implement the CalculatorServlet and the CalculatorServletJNDI in the lab-05-web module. You can find the pom.xml file and the java files in the folder lab05/files/lab-05-web.

5 minutes

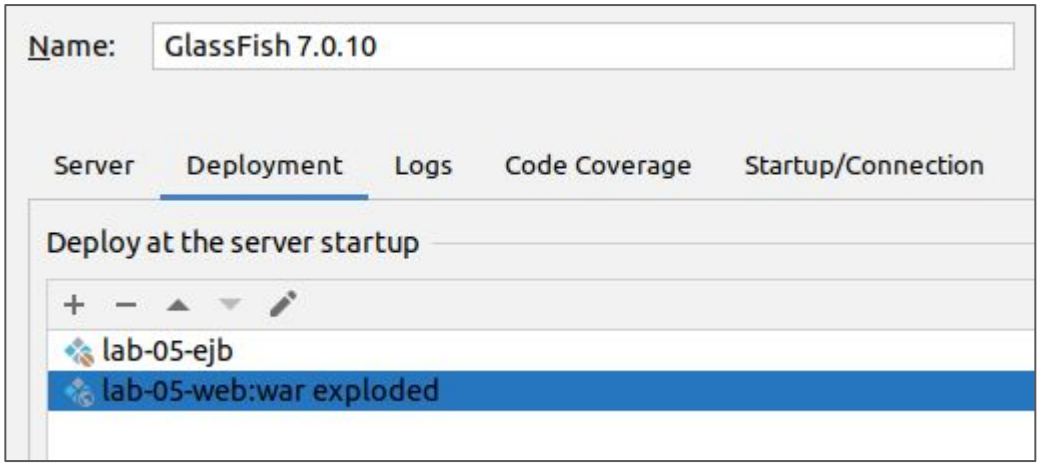
Step 5: deploying our EJB & Web Application in Glassfish (1)

We have to configure properly our **“EJB Application”**. To do so, go to **File -> Project Structure** and add the compile output of the **lab-05-ejb-interfaces** project to your artifact by double clicking on it. As a result, you should have set the configuration shown in the next image:



Step 5: deploying our EJB & Web Application in Glassfish (2)

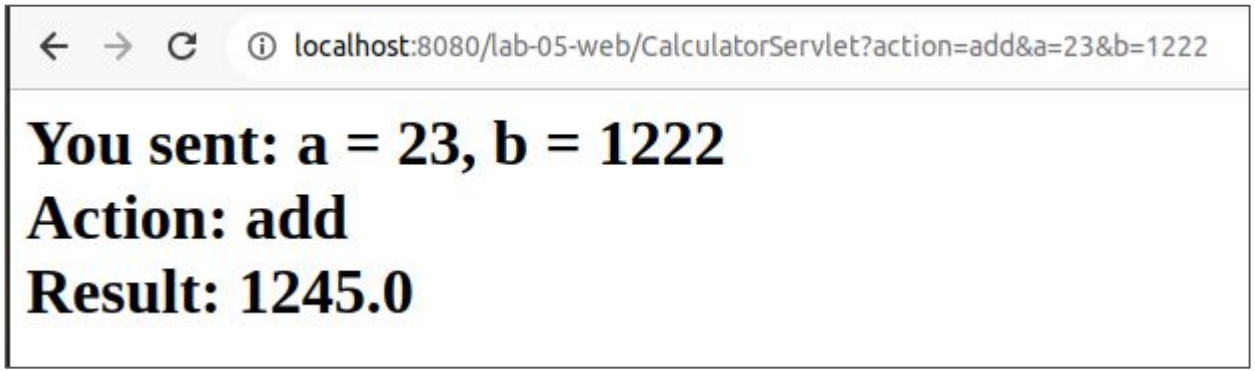
Finally, create a configuration to run your projects into Glassfish (as we did in our previous hands-on sessions). **Note that the order of the artifacts to deploy is important**: the Web Application fetch EJBs during the startup process so the EJB Application must be deployed first.



Step 5: deploying our EJB & Web Application in Glassfish (2)

Finally, let's test our application:

http://localhost:8080/lab-05-web/CalculatorServlet?action=add&a=23&b=1222



Exercise 07: Running a JakartaEE application

Please, follow the instructions described in the previous slides to run the Calculator Web application + Calculator EJB in Glassfish. For this exercise you have to perform the deployment from IntelliJ IDE.

Exercise 08: FortuneCookieServlet

For this exercise you have to create a servlet FortuneCookieServlet. This servlet will return a fortune cookie quote obtained from an EJB in the response. Everytime a quote is obtained a random one is picked up from a list of quotes.

Exercise 09: BeersServiceEJB

In lab03, exercise 3, it was requested to implement a BeersRESTAPI servlet that allows incoming request filtering stored beers data in a JSON file. It is required to migrate the search beers functionality to an EJB in another Maven project. Once this migration is done, you have to deploy this new project in Glassfish and get a reference of this EJB in the BeersRESTAPI servlet.

Exercise 10: Shopping Cart EJB

In our lab04 web application, it was requested to implement a shopping cart functionality which stores the information in the user session. For this exercise, it is requested to store the information of the shopping cart into an EJB. It is important to choose the correct EJB type to be used for this exercise/functionality.

References

- https://projects.eclipse.org/projects/ee4j