



UNIVERSITÀ DI PISA

AI-DII

Hands-on session 6: JMS

Distributed Systems and Middleware Technologies

2023/2024

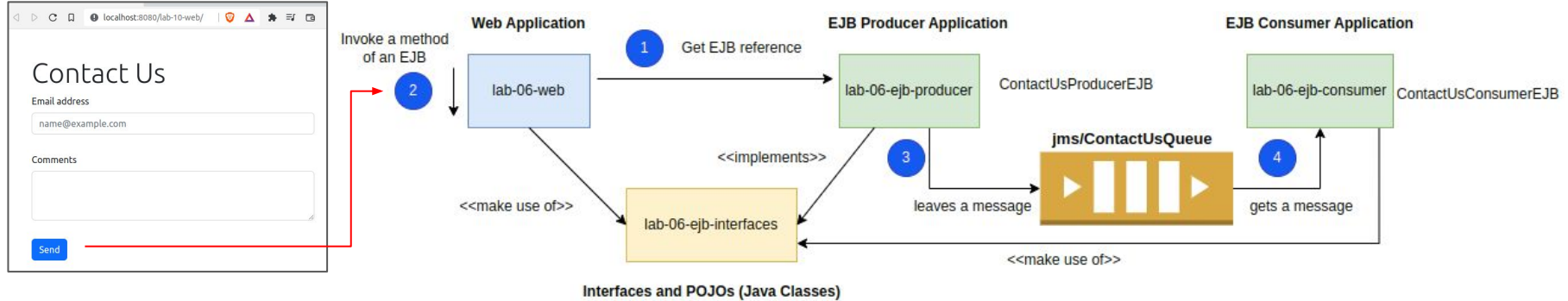
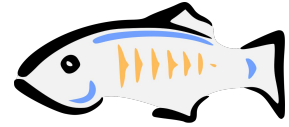
By José Luis Corcuera Bárcena

Agenda

1. Java Enterprise Application - JMS Project
2. Step 1: creating a queue in Glassfish
3. Step 2: defining the interfaces - pojos project
4. Step 3: defining the EJB Consumer Application project
5. Step 4: defining the EJB Producer Application project
6. Step 5: defining the Web Application project
7. Step 6: deployment in Glassfish
8. Exercises



Java Enterprise Application Project



Projects description:

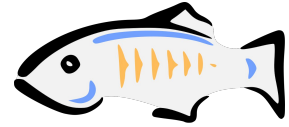
lab-06-web: It defines JSP, static resources, Servlets, etc.

lab-06-ejb-producer: It defines EJB (Our Business Logic). An EJB in this project will leave messages into a queue.

lab-06-ejb-interfaces: It defines the method signatures and data types to be used in the EJBs definition.

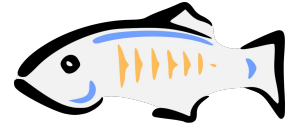
lab-06-ejb-consumer: An EJB in this project receives messages asynchronously from a queue.

Exercise 01: Project creation



For this exercise, it is required to define and create a project with some modules.
You are free to propose a project organization.

Step 1: creating a queue in Glassfish



Open the Glassfish Administration console and navigate to the Resources -> JMS Resources -> Destination Resources option in the navigation menu. The JMS Destination Resources will be displayed and there click on the button **New...** and fill in the information of the next page as it is shown in the screenshot below:

Eclipse GlassFish

Common Tasks

- Domain
 - server (Admin Server)
 - Clusters
 - Standalone Instances
 - Nodes
 - Applications
 - Lifecycle Modules
 - Monitoring Data
 - Resources
 - Concurrent Resources
 - Connectors
 - JDBC
 - JMS Resources
 - Connection Factories
 - Destination Resources**
 - jms/ContactUsQueue
 - JNDI
 - JavaMail Sessions
 - Resource Adapter Configs
 - Configurations

Edit JMS Destination Resource

Editing a Java Message Service (JMS) destination resource also modifies the associated admin object resource.

[Load Defaults](#)

JNDI Name: jms/ContactUsQueue

Physical Destination Name * contactusqueue
Destination name in the Message Queue broker. If the destination does not exist, it will be created automatically when needed.

Resource Type: * jakarta.jms.Queue

Deployment Order: 100
Specifies the loading order of the resource at server startup. Lower numbers are loaded first.

Description:

Status: ☒

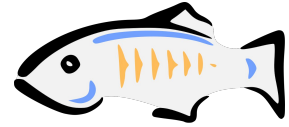
Additional Properties (0)

[Add Property](#) [Delete Properties](#)

Select	Name	Value	Description
No items found.			

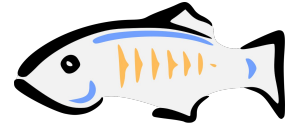
Do not use special characters (+, -, /, _, @, etc) in the Physical Destination Name field.

Exercise 02: JMS Queue Creation



For this exercise, it is required to create the Queue `ContactUsQueue` as it was shown in the previous slide.

Step 2: defining the interfaces - pojos project



As first step, we have to define a project with the interfaces and pojos (Java classes) to interact with the EJBs.

Project: lab-06-ejb-interfaces. After every code update in this project, run the “install” lifecycle.

Important: If you define POJOs, make sure they implement the interface Serializable.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
<modelVersion>4.0.0</modelVersion>
<groupId>it.unipi.dsmi.jakartaee</groupId>
<artifactId>lab-06-ejb-interfaces</artifactId>
<version>1.0.0-SNAPSHOT</version>
<packaging>jar</packaging>
<name>${artifactId}</name>
<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<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>
</project>
```

pom.xml

```
package it.unipi.dsmi.jakartaee.lab_06_ejb_interfaces.interfaces ;

import it.unipi.dsmi.jakartaee.lab_06_ejb_interfaces.dto.ContactUsDTO ;
import jakarta.ejb.Remote ;

@Remote
public interface ContactUsEJB {
    void processContactUsDTO (ContactUsDTO contactUsDTO);
}
```

ContactUsEJB.java

Step 3: defining the EJB Consumer Application project (1)



Next, we have to define a project which implements the interfaces defined in the step 1. **Project:**
lab-06-ejb-consumer

```
<groupId>it.unipi.dsmt.jakartaee</groupId>
<artifactId>lab-06-ejb-consumer</artifactId>
<version>1.0.0-SNAPSHOT</version>
<packaging>jar</packaging>
<name>${artifactId}</name>
<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<maven.compiler.source>11</maven.compiler.source>
<maven.compiler.target>11</maven.compiler.target>
</properties>
<dependencies>
  <dependency>
    <groupId>it.unipi.dsmt.jakartaee</groupId>
    <artifactId>lab-06-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 pom.xml

Step 3: defining the EJB Consumer Application project (2)



Next, we have to define a project receives messages from the defined Queue. **Project:**
lab-06-ejb-consumer

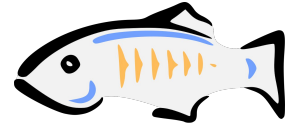
```
package it.unipi.dsmt.jakartaee.lab_06_ejb.consumer;

import it.unipi.dsmt.jakartaee.lab_06_ejb_interfaces.dto.ContactUsDTO;
import jakarta.ejb.ActivationConfigProperty;
import jakarta.ejb.MessageDriven;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageListener;

@MessageDriven(name = "ConsumerEJB",
    activationConfig = {
        @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue = "jms/ContactUsQueue"),
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "jakarta.jms.Queue")
    })
public class ContactUsConsumerEJB implements MessageListener {
    @Override
    public void onMessage(Message message) {
        ContactUsDTO contactUsDTO = null;
        try {
            contactUsDTO = message.getBody(ContactUsDTO.class);
            System.out.println("Message on consumer: " + contactUsDTO);
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

ContactUsConsumerEJB.java

Exercise 03: EJB consumer implementation and deployment



For this exercise, it is required to implement and deploy the lab-06-ejb-consumer module in Glassfish.

Step 4: defining the EJB Producer Application project (1)



Next, we have to define a project which implements the interfaces defined in the step 1. **Project:**
lab-06-ejb-producer

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>it.unipi.dsmi.jakartaee</groupId>
  <artifactId>lab-06-ejb-producer</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>${artifactId}</name>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>it.unipi.dsmi.jakartaee</groupId>
      <artifactId>lab-06-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>
</project>
```

pom.xml

```
@Stateless(mappedName = "ContactUsProducerEJB" )
public class ContactUsProducerEJB implements ContactUsEJB {
    static final String QC_FACTORY_NAME = "jms/ defaultConnectionFactory" ;
    static final String QUEUE_NAME = "jms/ContactUsQueue" ;
    private Queue queue ;
    private JMSContext jmsContext ;

    public ContactUsProducerEJB () {
        try{
            Context ic = new InitialContext();
            queue = (Queue) ic.lookup( QUEUE_NAME );
            QueueConnectionFactory qcf =
            (QueueConnectionFactory ) ic.lookup( QC_FACTORY_NAME );
            jmsContext = qcf.createContext();
        }
        catch (NamingException e) {
            System.err.println( "Unable to initialize ContactUsProducerEJB EJB." );
            e.printStackTrace();
        }
    }

    @Override
    public void processContactUsDTO (ContactUsDTO contactUsDTO) {
        System.out.println( "call ContactUsProducerEJB.processContactUsDTO" );
        jmsContext.createProducer().send( queue, contactUsDTO);
    }
}
```

Piece of ContactUsProducerEJB.java

Step 4: defining the EJB Producer Application project (2)



From JMS 2.0 we can simplified the configuration by injecting a JMSContext instance. This is a new object that combines Connection and Session capabilities. **Project: lab-06-ejb-producer**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>it.unipi.dsmi.jakartaee</groupId>
  <artifactId>lab-06-ejb-producer</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>${artifactId}</name>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>it.unipi.dsmi.jakartaee</groupId>
      <artifactId>lab-06-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>
</project>
```

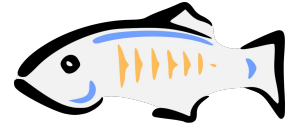
pom.xml

```
@Stateless(mappedName = "ContactUsProducerEJBV2")
public class ContactUsProducerEJBV2 implements ContactUsEJB {

    @Resource(lookup = "jms/ContactUsQueue")
    private Queue queue;
    @Inject
    private JMSContext jmsContext;
    @Override
    public void processContactUsDTQ(ContactUsDTO contactUsDTO) {
        System.out.println("call ContactUsProducerEJBV2.processContactUsDTQ");
        jmsContext.createProducer().send(queue, contactUsDTO);
    }
}
```

Piece of ContactUsProducerEJBV2.java

Exercise 04: EJB producer implementation and deployment



For this exercise, it is required to implement and deploy the lab-06-ejb-producer module in Glassfish.

Step 5: defining the Web Application project



In our Web Application, EJBs references are obtained by using the **@EJB** annotation. **Project: lab-06-web**

```
<groupId>it.unipi.dsmi.jakartaee</ groupId>
<artifactId>lab-06-web</ artifactId>
<packaging>war</ packaging>
<version>1.0.0-SNAPSHOT</ version>
<properties>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
</properties>
<name>${artifactId}</ name>
<dependencies>
  <dependency>
    <groupId>it.unipi.dsmi.jakartaee</ groupId>
    <artifactId>lab-06-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 pom.xml

```
@WebServlet(name = "ContactUsServlet", value = "/ContactUsServlet")
public class ContactUsServlet extends HttpServlet {
    @EJB(mappedName = "ContactUsProducerEJBV2")
    private ContactUsEJB contactUsEJB;

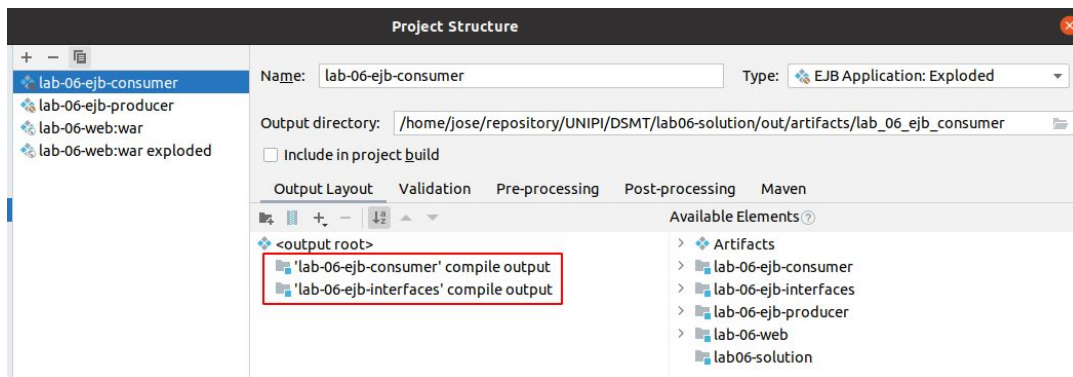
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String email = request.getParameter("email");
        String comments = request.getParameter("comments");
        ContactUsDTO dto = new ContactUsDTO();
        dto.setEmail(email);
        dto.setComments(comments);
        contactUsEJB.processContactUsDTO(dto);
        response.sendRedirect("contact-us-form-jms.jsp");
    }
}
```

ContactUsServlet.java

Step 6: Deployment in Glassfish (1)

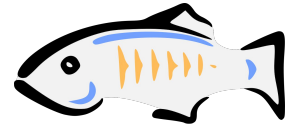


We have to configure properly our “**EJB Applications**”. To do so, go to **File -> Project Structure** and add the compile output of the **lab-06-ejb-interfaces** project to your artifact by double clicking on it. As a result, you should have set the configuration shown in the next image:

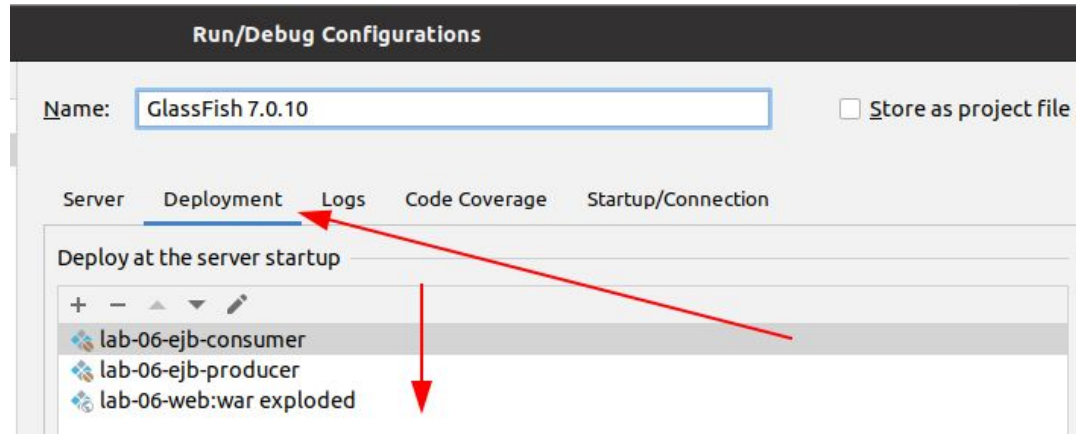


Repeat the same steps for the **lab-06-ejb-producer**.

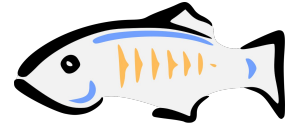
Step 6: Deployment in Glassfish (2)



There exist a dependency in during the initialization of the projects. Remember that the **lab-06-web** application needs to inject some EJB references so the **lab-06-ejb-producer** project needs to be deployed first.

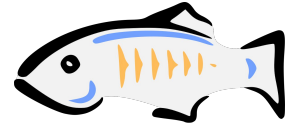


Exercise 05: Web App implementation and deployment



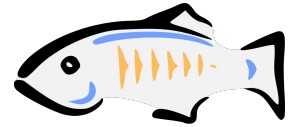
For this exercise, it is required to implement and deploy the lab-06-web module in Glassfish.

Exercise 06: Fetching remote EJB



For this exercise, let's deploy the lab-06-web application into an Apache Tomcat server. Make the necessary changes to fetch the ContactUsProducerEJB EJB reference.

Exercise 07: Shopping Cart Checkout



An exercise in our previous lab session was to implement the shopping cart functionality. In the “**View Cart**” page, add a checkout button. When this button is pressed, the information of the products selected in the shopping cart are sent to an **EJB** which will put them into a **queue**. Later, another **EJB** will pick them up and prints into the console the items that was left into the queue.

References

- <https://jakarta.ee/specifications/messaging/3.0/jakarta-messaging-spec-3.0.html>
- <http://devdoc.net/javaxe/JavaEE-7u2/docs/javaee-tutorial/doc/jms-examples002.htm>
- <https://www.javacodegeeks.com/2013/05/java-ee-7-jms-2-0-with-glassfish-v4.html>
- <https://stackoverflow.com/questions/7443306/javaee-6-javax-naming-namealreadyboundexception-use-rebind-to-override>
- <https://medium.com/platform-engineer/web-api-design-35df8167460>
- <https://blogs.oracle.com/javamagazine/post/how-to-build-applications-with-the-websocket-api-for-java-ee-and-jakarta-ee>
- <https://jakarta.ee/specifications/websocket/2.0/websocket-spec-2.0.html>