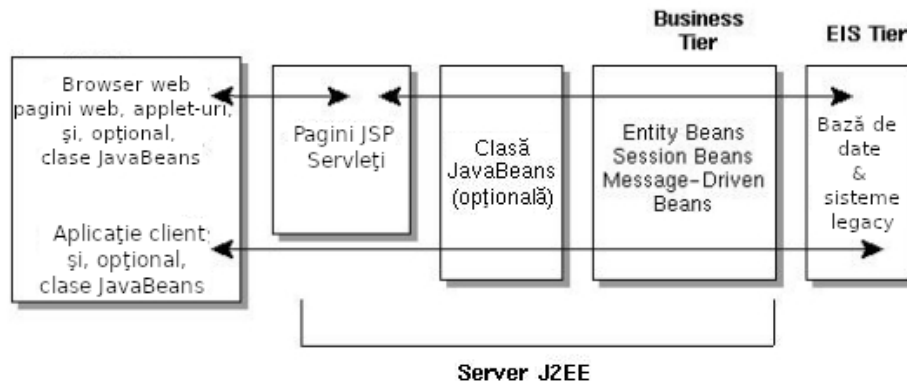


Laboratorul 2

Dezvoltarea componentelor de *business* într-o aplicație pentru întreprindere (JEE)

Introducere și arhitectura JEE



În laboratorul 1 s-a pus accent pe o arhitectură simplificată JEE, ce conține componente ce fac parte din **web tier**, respectiv componente **client** (browser-ul web sau utilitar capabil de a trimite cereri HTTP - **curl**).

În acest laborator, veți lucra la nivel de **business tier**, punând astfel accent pe tipurile de componente *business* din platforma JEE:

- **session beans** - de 3 tipuri:
 - **stateless session beans** → nu mențin o legătură **client** ↔ **bean**
 - **stateful session beans** → mențin legătura cu clientul apelant (fiecare client cu *stateful session bean*-ul lui)
 - **singleton session beans** → o singură instanță disponibilă la nivel de server *enterprise*
- **entity beans** → încapsulează funcționalitate, respectiv maparea datelor dintr-o bază de date sub formă de obiecte (învechite începând de la EJB 3.0, înlocuite de entitățile de tip **JPA entity** din Java Persistence API)
- **message-driven beans** → permit procesarea mesajelor în mod asincron

Aplicație JEE completă

În laboratorul 1, ați creat un proiect JEE minimal, de tip **Web Application**, cu împachetare **WAR** (Web ARchive). În acest laborator, deoarece urmează să utilizați componentele de *business tier*, veți învăța cum se creează un proiect JEE complet, de tip **Enterprise Application**, cu împachetare **EAR** (Enterprise ARchive) din IntelliJ IDEA Community și cum se adaugă dependențele necesare.

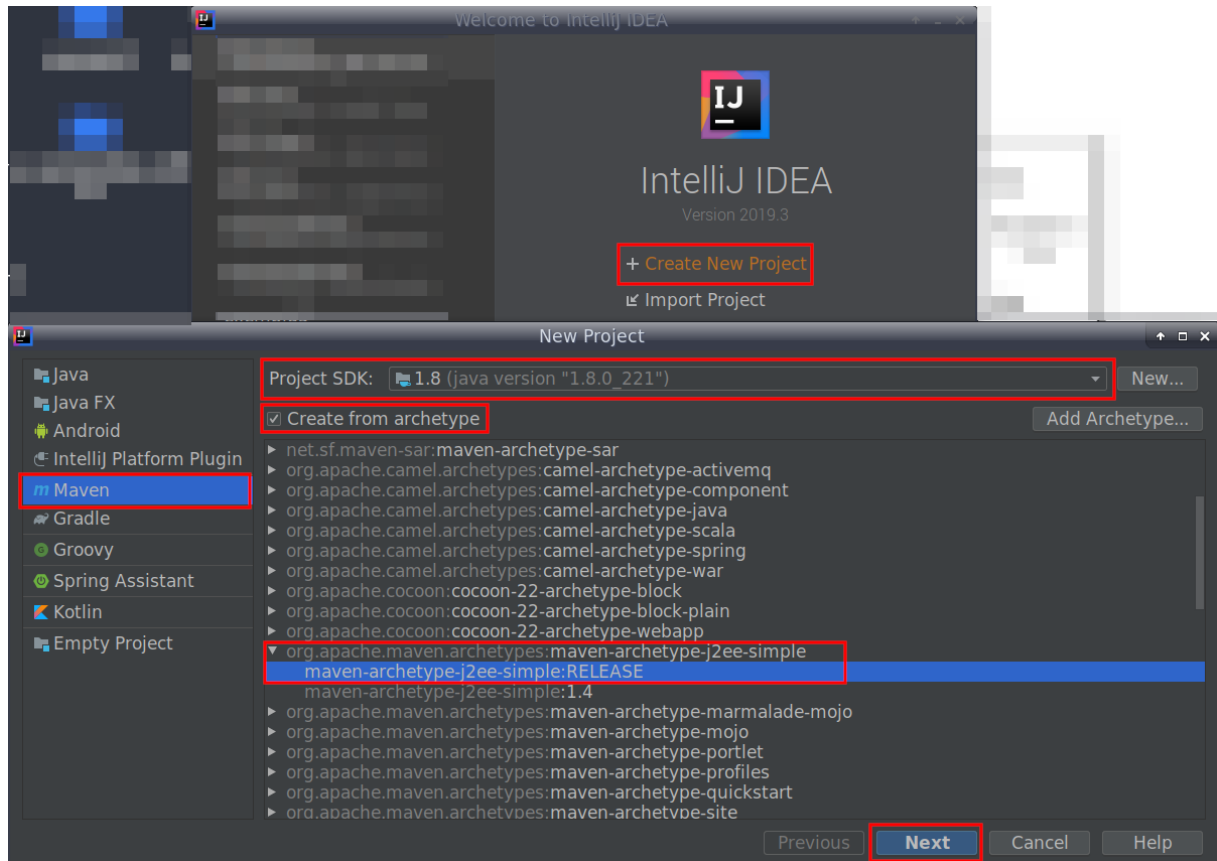
După terminarea orei de laborator, deschideți consola Glassfish: <http://localhost:4848>, dați click pe Applications în partea stângă, bifați tot în afară de cargocpc și apăsați pe butonul Undeploy.

1.1. Creare și configurare proiect JEE complet folosind IntelliJ IDEA Community

1.1.1. Creare proiect IntelliJ

Deschideți IntelliJ IDEA Community, iar în meniul din partea dreaptă alegeți „**Create new project**”.

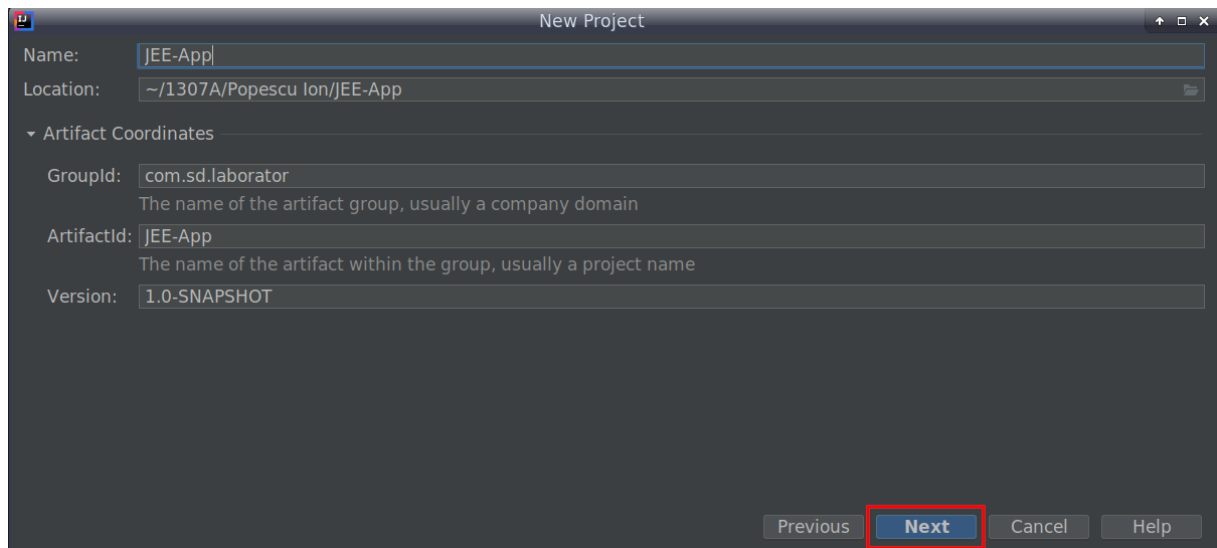
În fereastra de selecție a tipului de proiect, alegeți „**Maven**” în partea stângă, apoi selectați versiunea de **Java SDK 1.8**. Bifați „**Create from archetype**”, iar din lista de arhetipuri disponibile, expandați **org.apache.maven.archetypes:maven-archetype-j2ee-simple** și selectați **maven-archetype-j2ee-simple:RELEASE**. Click pe „**Next**”.



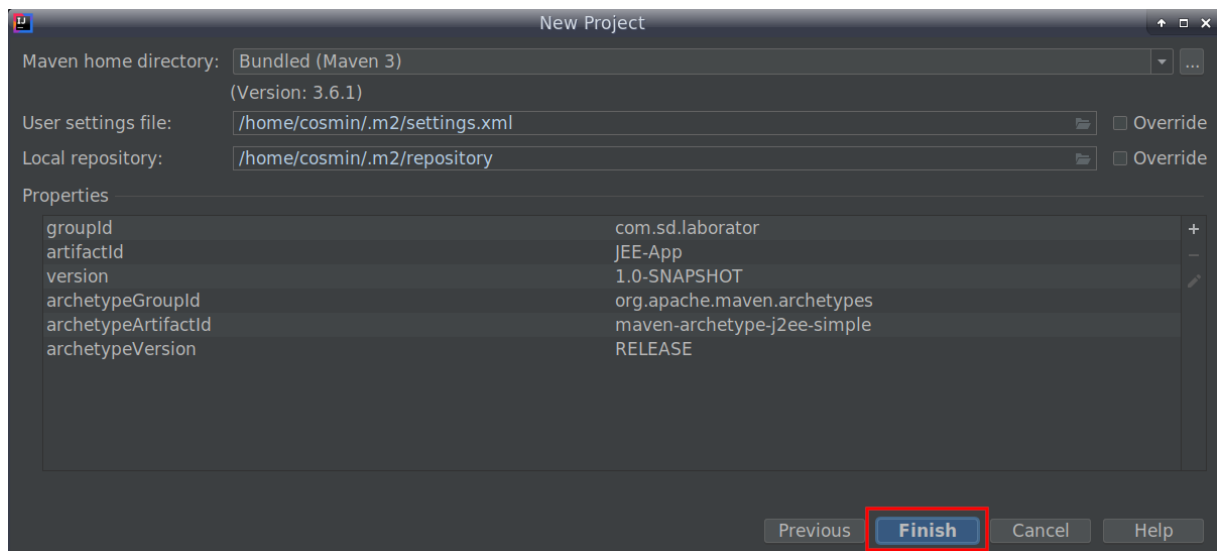
În continuare, se alege numele și locația proiectului pe disc, precum și detaliile artefactului EAR rezultat. Secțiunea „**Artifact Coordinates**” poate fi lăsată cu valorile implicite, sau puteți completa, dacă doriți, **GroupId**-ul cu o valoare personalizată, cum ar fi **com.sd.laborator**.

În acest exemplu, proiectul se va numi „**JEE-App**”, iar locația va fi **~/1307A/Popescu Ion/JEE-App**.

Click pe „**Next**” după completarea datelor menționate.



În următoarea fereastră, se lasă totul neschimbat și se apasă „**Finish**”.



Așteptați ca Maven să termine de generat structura proiectului și să aducă primele dependențe definite în configurația arhetipului.

1.1.2. Configurare proiect Maven

Proiectul Maven generat de arhetipul pentru JEE este organizat în module, fiecare modul conținând anumite tipuri de componente, după cum urmează:

- **ear** → generează artefactul EAR ce încapsulează toate celelalte module ale proiectului sub formă de arhivă **Enterprise ARchive**.
- **ejbs** → conține codul de *business* al claselor *Enterprise Java Beans*. Aici veți pune clasele EJB, organizate eventual în pachete.
- **primary-source** → conține clase adiționale utilizate în proiect (clase care nu reprezintă neapărat componente specifice JEE)
- **projects** → conține eventuale subproiecte ale proiectului JEE de bază. Are deja adăugat un schelet de subproiect denumit **logging**.
- **servlets** → conține un submodul numit **servlet**, care reprezintă punctul de intrare al componentei web (nivelul *web tier*). Aici puteți adăuga clase servlet, pagini JSP, etc.

Calea implicită de acces a folder-ului rădăcină a componentei web este `/servlet`.
 Așadar, pentru a vizualiza pagina `index.jsp` creată automat, după încărcarea aplicației, se accesează URL-ul: <http://localhost:8080/servlet/>.

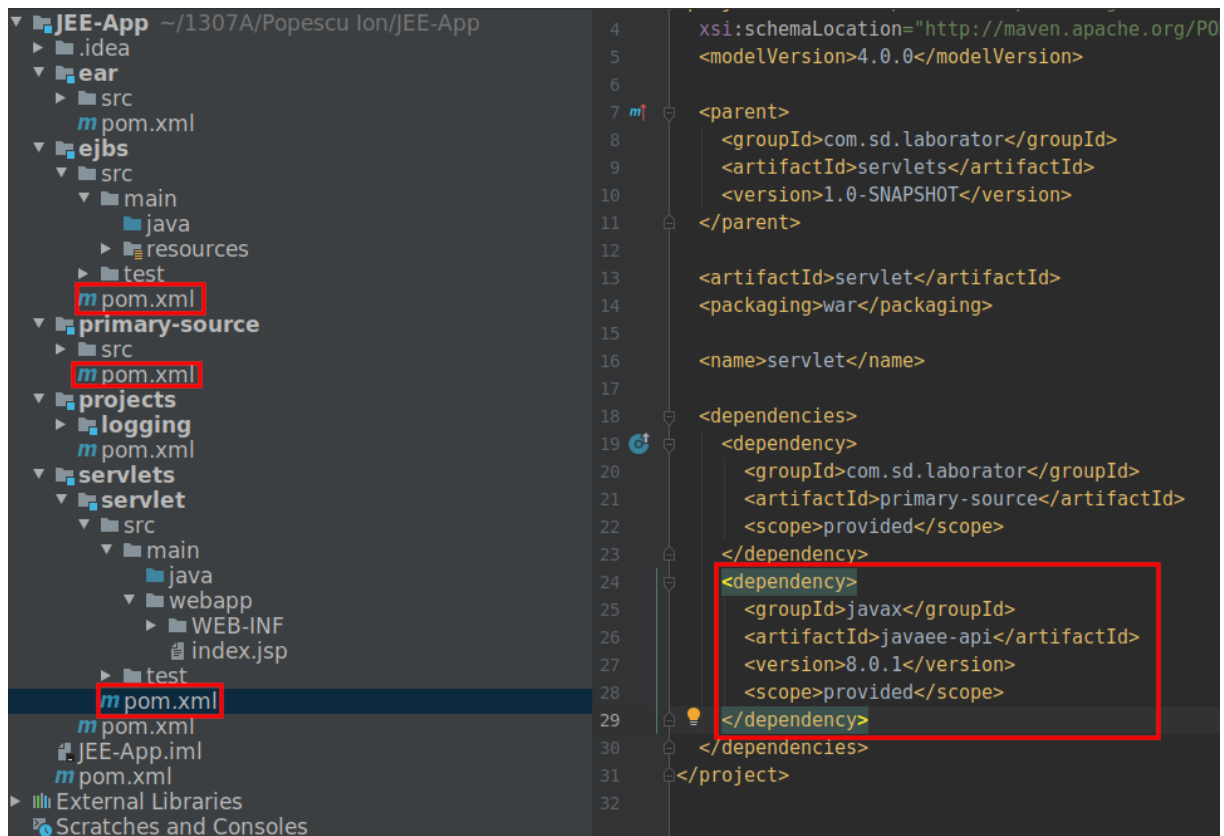
Structura de proiect arată ca în figură:



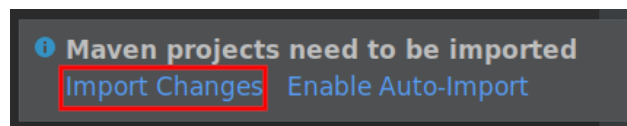
Observați că fiecare modul are fișierul său de configurare `pom.xml` (**Project Object Model**), deoarece se pot declara dependențe între modulele componente, respectiv, pentru fiecare modul în parte se pot face configurări personalizate, se pot declara dependențe și folosi *plugin-uri* personalizate etc.

Adăugați dependența **JavaEE API** (<https://mvnrepository.com/artifact/javax/javaee-api/8.0>) în fiecare fișier `pom.xml` al modulelor `ejbs`, `primary-source` și `servlet` (**atenție, nu servlets!**). Așadar, ca subordonat al `tag`-ului `<dependencies>` din fișierele `pom.xml`, adăugați următorul element:

```
<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-api</artifactId>
  <version>8.0.1</version>
  <scope>provided</scope>
</dependency>
```



După modificările făcute în fișierele `pom.xml`, nu uitați să sincronizați modificările, apăsând „**Import changes**” când IntelliJ vă cere acest lucru:



1.1.3. Completare descriptori XML lipsă

Arhetipul din care proiectul Maven a fost creat nu adaugă conținut în descriptorii XML ai componentelor web, respectiv EJB. Aceștia trebuie adăugați manual, astfel:

Dechideți `servlets/servlet/src/main/webapp/WEB-INF/web.xml` și adăugați următorul conținut:

```

<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Web Component App</display-name>
</web-app>

```

Apoi deschideți `ejbs/src/main/resources/META-INF/ejb-jar.xml` și adăugați următorul conținut:

```

<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"

```

```

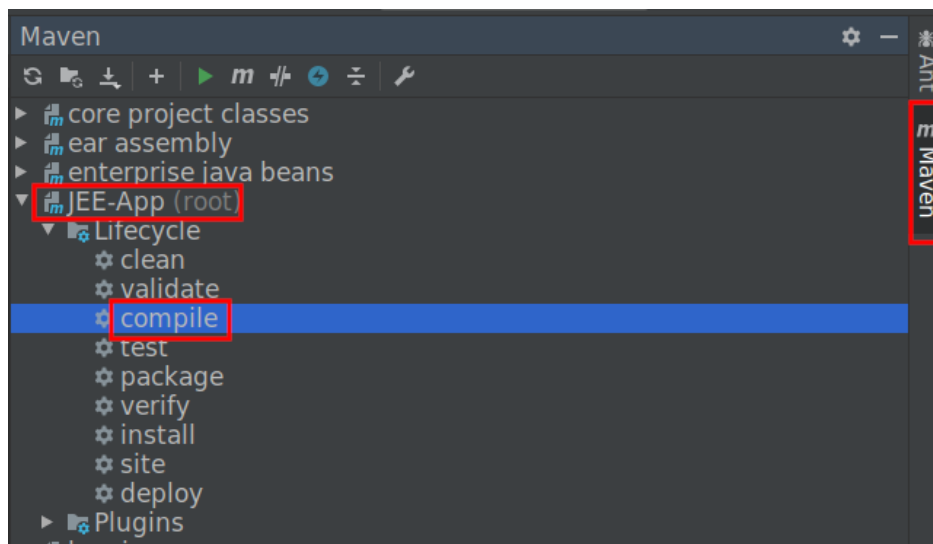
        version="3.0">
</ejb-jar>

```

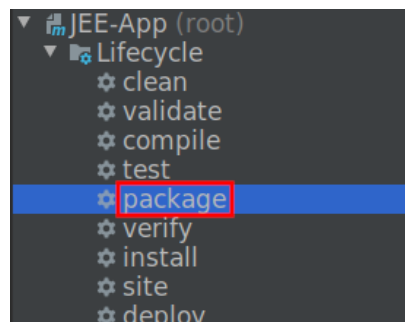
1.2. Compilarea și împachetarea aplicației JEE

Operațiile care pot fi făcute asupra proiectului sunt disponibile sub formă de *Maven lifecycles* în partea dreaptă a ferestrei IntelliJ, în conținutul panoului **Maven**.

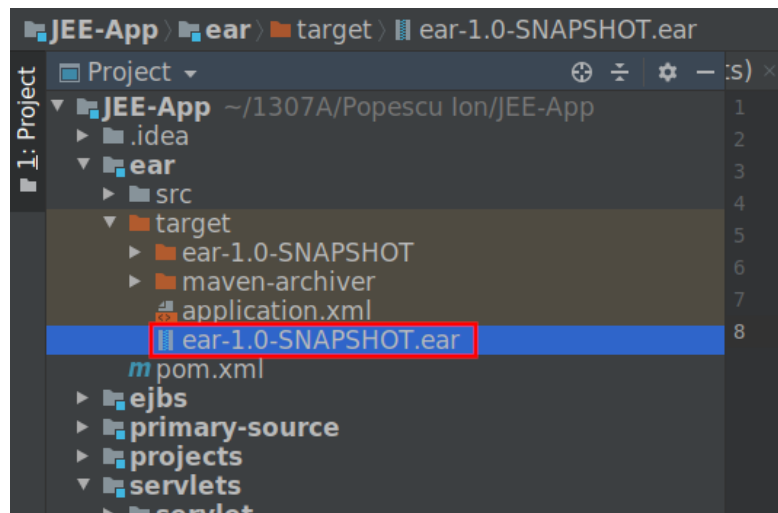
Pentru a compila proiectul, se expandează panoul Maven și se utilizează *lifecycle*-ul **compile** al modului Maven care cuprinde întregul proiect, adică cel denumit exact ca și proiectul IntelliJ: **<nume_proiect> (root)**.



Tot din același modul Maven se poate împacheta structura aplicației rezultate în urma compilării (conținutul folder-ului **target**) folosind *lifecycle*-ul **package**.

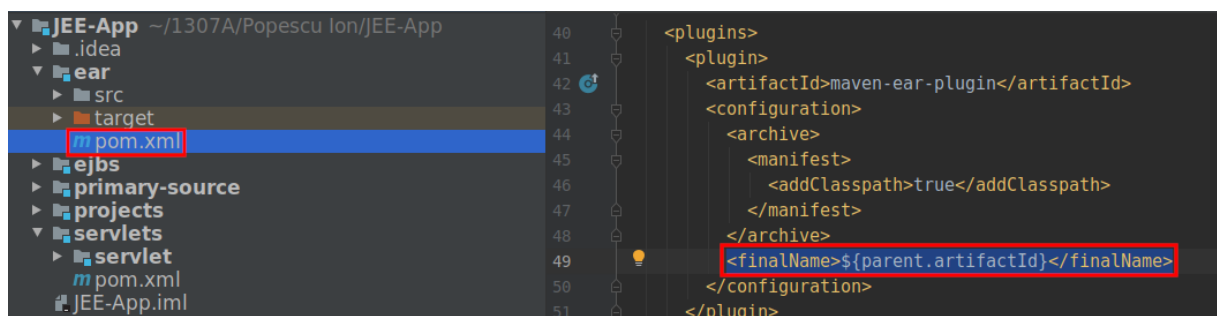


Acest pas va genera un fișier **EAR** (Enterprise ARchive) ca subordonat al folder-ului **target** din modulul **ear**, vizibil în structura de proiect din partea stângă a ferestrei IntelliJ. Artefactul va fi denumit, în mod implicit, sub forma **ear-*<versiune>*.ear**.



Numele artefactului rezultat este stabilit în mod implicit conform regulii: `<nume_modul_ear>-<versiune>.ear`. Pentru a vă ușura munca atunci când veți încărca aplicația pe server, configurați modulul ear astfel încât să exporte artefactul EAR sub un nume mai „prietenos”, și anume cel al proiectului părinte. Deschideți `ear/pom.xml` și adăugați următoarea configurare ca și subordonat al `tag`-ului `<configuration>` pentru `plugin`-ul `maven-ear-plugin`:

```
<finalName>${parent.artifactId}</finalName>
```



După ce împachetați din nou aplicația (cu *lifecycle*-ul **package**), veți observa că rezultă un artefact EAR sub numele de **JEE-App.ear** (mult mai sugestiv ca cel vechi, implicit). Cel vechi poate fi șters.

1.3. Curățarea proiectului

Tot din panoul Maven se poate curăța folder-ul cu fișiere compilate prin utilizarea *lifecycle*-ului **clean**. Acesta va șterge tot conținutul folder-elor **target** al modulelor componente. Dacă se dorește curățarea doar a anumitor module, se execută *lifecycle*-ul **clean** din secțiunea corespunzătoare acestora din panoul Maven.

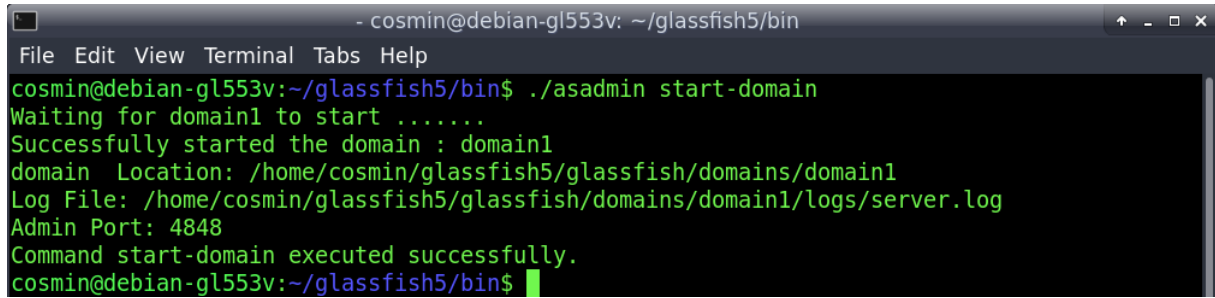
1.4. Pornirea server-ului GlassFish

Dintr-o sesiune de terminal, executați următoarea comandă:

```
<LOCAȚIE_SERVER_GLASSFISH>/bin/asadmin start-domain
```

De exemplu, în acest caz, server-ul fiind localizat în `/home/student/opt/glassfish5`, comanda este:

```
~/opt/glassfish5/bin/asadmin start-domain
```



```
cosmin@debian-gl553v: ~/glassfish5/bin
File Edit View Terminal Tabs Help
cosmin@debian-gl553v:~/glassfish5/bin$ ./asadmin start-domain
Waiting for domain1 to start .....
Successfully started the domain : domain1
Domain Location: /home/cosmin/glassfish5/glassfish/domains/domain1
Log File: /home/cosmin/glassfish5/glassfish/domains/domain1/logs/server.log
Admin Port: 4848
Command start-domain executed successfully.
cosmin@debian-gl553v:~/glassfish5/bin$
```

1.5. Oprirea server-ului GlassFish

Într-o sesiune de terminal, se execută comanda:

```
<LOCATIE_SERVER_GLASSFISH>/bin/asadmin stop-domain
```

În acest caz, ar fi:

```
~/opt/glassfish5/bin/asadmin stop-domain
```

1.6. Încărcarea proiectului (deploy) pe server-ul GlassFish

Deoarece s-a utilizat un arhetip de proiect JEE modular, încărcarea artefactului EAR pe server-ul de aplicații *enterprise* se poate face mai facil din linia de comandă, decât cu *plugin*-ul Cargo, utilizat în laboratorul 1.

Atenție: nu puteți încărca o aplicație în format EAR pe server-ul GlassFish dacă nu există cel puțin un tip de *Enterprise Bean* creat și descris în `ejb-jar.xml`. Deci, nu veți putea utiliza efectiv comenzile descrise în cele ce urmează decât după ce adăugați *bean-uri*.

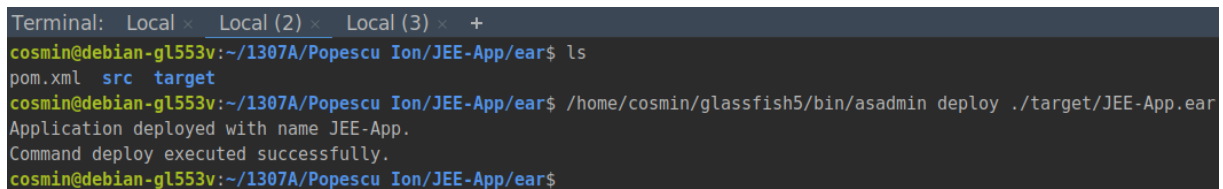
După ce ați compilat și împachetat aplicația sub formă de artefact EAR, click dreapta pe folder-ul **ear** → **Open in Terminal**. Se va deschide o sesiune de terminal în partea de jos a ferestrei IntelliJ.

Încărcarea proiectului (prima dată) se face cu următoarea comandă:

```
<LOCATIE_SERVER_GLASSFISH>/bin/asadmin deploy
/CALE/CĂTRE/ARTEFACT/<NUME_ARTEFACT>.ear
```

De exemplu:

```
~/opt/glassfish5/bin/asadmin deploy ./target/JEE-App.ear
```



```
Terminal: Local x Local (2) x Local (3) x +
cosmin@debian-gl553v:~/1307A/Popescu Ion/JEE-App/ear$ ls
pom.xml src target
cosmin@debian-gl553v:~/1307A/Popescu Ion/JEE-App/ear$ /home/cosmin/glassfish5/bin/asadmin deploy ./target/JEE-App.ear
Application deployed with name JEE-App.
Command deploy executed successfully.
cosmin@debian-gl553v:~/1307A/Popescu Ion/JEE-App/ear$
```

1.7. Ștergerea proiectului (undeploy) de pe server-ul GlassFish

Dacă se dorește ca aplicația *enterprise* să fie ștearsă de pe server, se poate folosi comanda:

```
<LOCATIE_SERVER_GLASSFISH>/bin/asadmin undeploy <NUME_APLICAȚIE>
```

Exemplu:


```
~/opt/glassfish5/bin/asadmin undeploy JEE-App
```

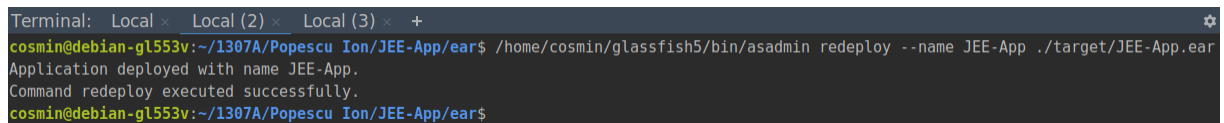
1.8. Reîncărcarea proiectului (redploy) pe server-ul GlassFish

După ce încărcăți pentru prima dată o aplicație JEE pe server, dacă actualizați fișierele sursă și doriți să vedeți modificările, trebuie să reîncărcați artefactul nou rezultat în urma împachetării. Deoarece artefactul are același nume după reîmpachetare (dacă nu îl modificați), server-ul nu vă permite să suprascrieți aplicația veche, decât folosind o comandă de **reîncărcare (redploy)**:

```
<LOCAȚIE_SERVER_GLASSFISH>/bin/asadmin redploy <NUME_APLICAȚIE>  
/CALE/CĂTRE/ARTEFACT/<NUME_ARTEFACT>.ear
```

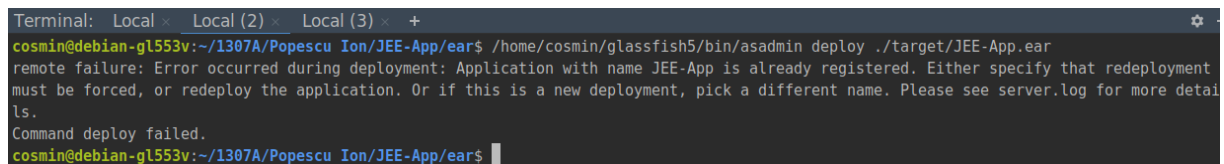
Exemplu:

```
~/opt/glassfish5/bin/asadmin redploy --name JEE-App ./target/JEE-App.ear
```



```
Terminal: Local x Local (2) x Local (3) x +  
cosmin@debian-g1553v:~/1307A/Popescu Ion/JEE-App/ear$ /home/cosmin/glassfish5/bin/asadmin redploy --name JEE-App ./target/JEE-App.ear  
Application deployed with name JEE-App.  
Command redeploy executed successfully.  
cosmin@debian-g1553v:~/1307A/Popescu Ion/JEE-App/ear$
```

Dacă încercați să folosiți comanda **deploy** pentru un artefact deja existent (cu același nume), veți primi o eroare de tipul:



```
Terminal: Local x Local (2) x Local (3) x +  
cosmin@debian-g1553v:~/1307A/Popescu Ion/JEE-App/ear$ /home/cosmin/glassfish5/bin/asadmin deploy ./target/JEE-App.ear  
remote failure: Error occurred during deployment: Application with name JEE-App is already registered. Either specify that redeployment must be forced, or redeploy the application. Or if this is a new deployment, pick a different name. Please see server.log for more details.  
Command deploy failed.  
cosmin@debian-g1553v:~/1307A/Popescu Ion/JEE-App/ear$
```

Stateless Session Beans

Stateless Session Bean-ul este un *enterprise bean* care, de obicei, efectuează operații independente de clientul apelant (nu creează nicio legătură bean ↔ client, nu păstrează starea conversației dintre cele 2 părți participante).

Pentru a exemplifica acest tip de *enterprise bean*, veți expune clientului o interfață simplă prin care poate prelua data curentă de la server și prin care poate aduna 2 numere întregi. Această interfață trebuie cunoscută atât de server-ul de aplicații *enterprise* (ca să știe ce metode trebuie să expună în momentul încărcării aplicației), cât și de clientul apelant (pentru ca el să știe ce metode sunt disponibile pentru apel).

Folosiți în continuare proiectul JEE creat în pașii anteriori.

1.9. Codul de business

Adăugați un pachet nou în folder-ul **ejbs/src/main/java**, denumit **interfaces**. În pachetul respectiv, creați o interfață Java numită **StatelessSessionBeanRemote**, cu următorul conținut:

```
package interfaces;  
  
public interface StatelessSessionBeanRemote {  
    String getCurrentTime();  
}
```

```
Integer addNumbers(Integer a, Integer b);
}
```

Acum, veți crea implementarea conform interfeței, corpul efectiv al *stateless session bean*-ului. În folder-ul **ejbs/src/main/java**, creați un pachet denumit **ejb** (Enterprise Java Beans). În pachetul respectiv, creați o clasă Java denumită **StatelessSessionBeanImpl**, cu următorul conținut:

```
package ejb;

import interfaces.StatelessSessionBeanRemote;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Date;

public class StatelessSessionBeanImpl implements
StatelessSessionBeanRemote, Serializable {
    public StatelessSessionBeanImpl() {
        System.out.println("[Glassfish] S-a instanțiat un stateless
session bean: " +
            StatelessSessionBeanImpl.class.getName());
    }

    public String getCurrentTime() {
        System.out.println("[Glassfish] S-a apelat metoda
getCurrentTime()");
        Date date = new Date();
        SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy
HH:mm:ss");
        return formatter.format(date);
    }

    public Integer addNumbers(Integer a, Integer b) {
        System.out.println("[Glassfish] S-a apelat metoda addNumbers("
+ a + ", " + b + ")");
        return a + b;
    }
}
```

Clasa ce reprezintă acest bean trebuie să implementeze interfața **Serializable**, deoarece codul va fi apelat prin RMI (**Remote Method Execution**) de la o aplicație client complet decuplată de server, și atunci entitățile conținute de acea clasă (inclusiv clasa însăși) trebuie să fie serializabile.

Atenție: nu folosiți tipuri primitive de date în enterprise beans care vor fi apelate prin RMI (bean-uri nelocale)! Tipurile primitive de date (int, char, double etc.) nu sunt serializabile. Folosiți, în schimb, clasele lor corespondente Java: Integer, Char, Double etc. De aceea, pentru metoda addNumbers, s-au folosit parametri de tip Integer, și nu int, atât în interfață, cât și în implementare.

Dacă bean-urile *enterprise* conțin tipuri primitive de date, clientul apelant va primi o eroare de acest tip:

```

Run: JEEClient
/usr/lib/jvm/oracle-java8-jdk-amd64/bin/java ...
Jan 04, 2020 7:17:01 PM com.sun.enterprise.v3.server.CommonClassLoaderServiceImpl findDerbyClient
INFO: Cannot find javaee client jar file, derby jdbc driver will not be available by default.
04-01-2020 19:17:03
Exception in thread "main" javax.ejb.EJBException: java.rmi.MarshalException: CORBA MARSHAL 1330446343 No; nested exception is:
org.omg.CORBA.MARSHAL: FINE: 00810007: Underflow in BufferManagerReadStream after last fragment in message vmcid: OMG minor code: 7 completed: No
at interfaces.StatelessSessionBeanRemote_Wrapper.addNumbers(interfaces/_StatelessSessionBeanRemote_Wrapper.java)
at JEEClient.main(JEEClient.java:13)
Caused by: java.rmi.MarshalException: CORBA MARSHAL 1330446343 No; nested exception is:
org.omg.CORBA.MARSHAL: FINE: 00810007: Underflow in BufferManagerReadStream after last fragment in message vmcid: OMG minor code: 7 completed: No
at com.sun.corba.ee.impl.javax.rmi.CORBA.Util.mapSystemException(Util.java:267)
at com.sun.corba.ee.impl.presentation.rmi.StubInvocationHandlerImpl.privateInvoke(StubInvocationHandlerImpl.java:201)
at com.sun.corba.ee.impl.presentation.rmi.StubInvocationHandlerImpl.invoke(StubInvocationHandlerImpl.java:140)
at com.sun.corba.ee.impl.presentation.rmi.codegen.CodegenStubBase.invoke(CodegenStubBase.java:226)
at interfaces._StatelessSessionBeanRemote_Remote_DynamicStub.addNumbers(interfaces/_StatelessSessionBeanRemote_Remote_DynamicStub.java)
... 2 more
Caused by: org.omg.CORBA.MARSHAL: FINE: 00810007: Underflow in BufferManagerReadStream after last fragment in message vmcid: OMG minor code: 7 completed: No
at com.sun.corba.ee.impl.encoding.BufferManagerReadStream.underflow(BufferManagerReadStream.java:122)
at com.sun.corba.ee.impl.encoding.CDRInputStream_1_1.grow(CDRInputStream_1_1.java:111)
at com.sun.corba.ee.impl.encoding.CDRInputStream_1_2.alignAndCheck(CDRInputStream_1_2.java:126)

```

În continuare, folosind descriptori XML, trebuie să îi specificăm server-ului de aplicații *enterprise* ce reprezintă clasa creată mai sus (un *bean stateless*), cum se expune aceasta clientului, unde poate găsi interfața pe care o implementează, respectiv unde găsește clasa cu implementarea. Așadar, în folder-ul `ejbs/src/main/resources/META-INF`, adăugați următorul conținut în fișierul `ejb-jar.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">
  <enterprise-beans>
    <session>
      <ejb-name>StatelessSessionBeanExample</ejb-name>
      <mapped-name>ssb-example</mapped-name>
      <business-
local>interfaces.StatelessSessionBeanRemote</business-local>
      <business-
remote>interfaces.StatelessSessionBeanRemote</business-remote>
      <ejb-class>ejb.StatelessSessionBeanImpl</ejb-class>
      <session-type>Stateless</session-type>
    </session>
  </enterprise-beans>
</ejb-jar>

```

În acest XML, se declară faptul că aplicația conține un *session bean* (tag-ul `<session>`), cu următoarele proprietăți:

- un nume descriptiv (**nu are legătură cu numele clasei!**): tag-ul `<ejb-name>`
- identificatorul sub care *bean*-ul este mapat în JNDI (*Java Naming and Directory Interface*): tag-ul `<mapped-name>`
- interfețele care sunt implementate de clasa ce reprezintă *bean*-ul (locală - **business-local**, respectiv de la distanță - **business-remote**), date sub forma `<pachet>.<nume_clasă>`.
- numele clasei care conține implementarea *bean*-ului, dat sub forma `<pachet>.<nume_clasă>`: tag-ul `<ejb-class>`
- tipul de *bean*: tag-ul `<session-type>`

Faceți următoarele operațiuni asupra proiectului: **clean** → **compile** → **package** → **deploy** **de la consolă** (nu uitați să porniți server-ul **GlassFish**, dacă nu este deja pornit).

Dacă totul a decurs bine, veți regăsi în consola de administrare GlassFish *bean*-ul creat printre celelalte componente deja existente pe server:

Modules and Components (5)					
Module Name	Engines	Component Name	Type	Action	
com.sd.laborator-ejbs-1.0-SNAPSHOT.jar	[ejb]	-----	-----		
com.sd.laborator-ejbs-1.0-SNAPSHOT.jar		StatelessSessionBeanExample	StatelessSessionBean		
com.sd.laborator-servlet-1.0-SNAPSHOT.war	[web]	-----	-----		
com.sd.laborator-servlet-1.0-SNAPSHOT.war		default	Servlet		
com.sd.laborator-servlet-1.0-SNAPSHOT.war		jsp	Servlet		

1.10. Extragerea numelui JNDI al bean-ului

Server-ul de aplicații enterprise expune bean-ul respectiv printr-un nume JNDI mapat conform a ceea ce s-a specificat în descriptorul XML (conținutul tag-ului **<mapped-name>**). Însă, un client care caută și utilizează *bean*-ul de pe server are nevoie de numele complet JNDI stabilit de GlassFish, conform specificațiilor proprii. Acest nume poate fi găsit în log-ul server-ului, disponibil la locația următoare:

<LOCAȚIE_SERVER_GLASSFISH>/glassfish/domains/domain1/logs/server.log

În acest caz, log-ul se află în locația:

/home/student/opt/glassfish5/glassfish/domains/domain1/logs/server.log

Deschideți log-ul cu un editor de text și căutați numele mapat *bean*-ului pe care l-ați creat (în acest caz, căutați „**ssb-example**”).

```

9993 [2020-01-02T18:41:25.807+0200] [glassfish 5.0] [INFO] [AS-EJB-00054] [javax.enterprise.ejb.container] [tid: _ThreadID=44
    _ThreadName=admin-listener(3)] [timeMillis: 1577983285807] [levelValue: 800] [[
9994   Portable JNDI names for EJB StatelessSessionBeanImpl: [java:global/JEE-Test/
    StatelessSessionBeanImpl!interfaces.StatelessSessionBeanRemote, java:global/JEE-Test/StatelessSessionBeanImpl]]]
9995
9996 [2020-01-02T18:41:25.807+0200] [glassfish 5.0] [INFO] [AS-EJB-00055] [javax.enterprise.ejb.container] [tid: _ThreadID=44
    _ThreadName=admin-listener(3)] [timeMillis: 1577983285807] [levelValue: 800] [[
9997   Glassfish-specific (Non-portable) JNDI names for EJB StatelessSessionBeanImpl: [ssb-example,
    ssb-example#interfaces.StatelessSessionBeanRemote]]]
9998
9999 [2020-01-02T18:41:25.845+0200] [glassfish 5.0] [INFO] [AS-WEB-GLUE-00172] [javax.enterprise.web] [tid: _ThreadID=44
    _ThreadName=admin-listener(3)] [timeMillis: 1577983285845] [levelValue: 800] [[
0000   Loading application [JEE-Test] at [/JEE-Test]]]
0001

```

Notați numele complet JNDI al bean-ului atribuit de server (**de tip Glassfish-specific, nu cel portabil!**), veți avea nevoie de el în aplicația client. În acest caz:

ssb-example#interfaces.StatelessSessionBeanRemote

La modul general, numele JNDI specific GlassFish al unui bean enterprise atribuit de de server este:

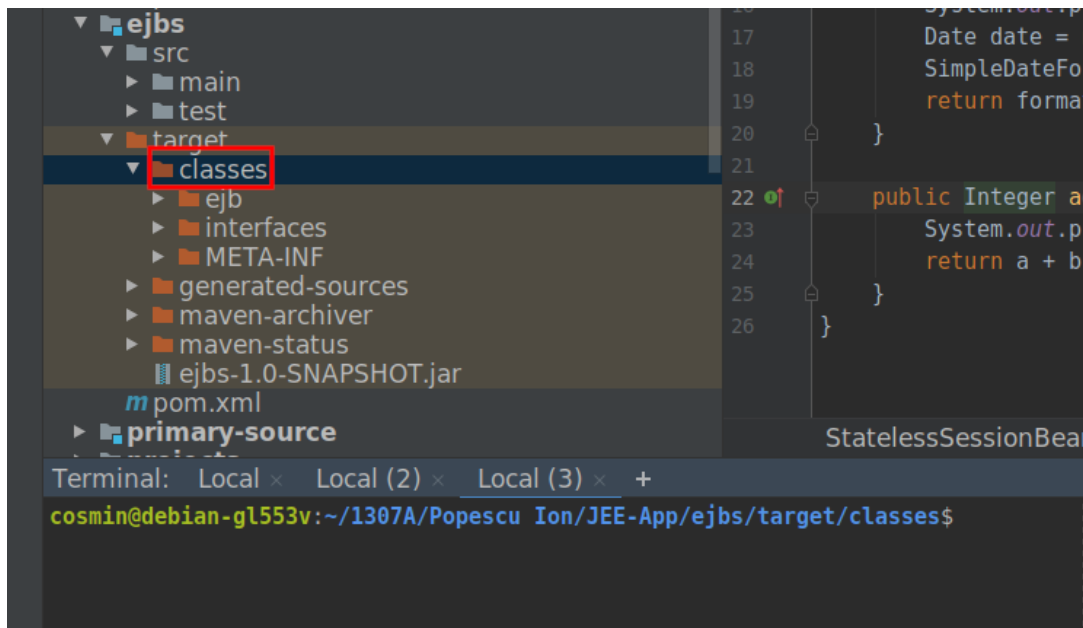
<NUME_MAPAT_ÎN_DESCRIPTOR>#<NUME_PACHET>.<NUME_INTERFAȚĂ>

1.11. Aplicație client

Clientul va fi, în acest caz, complet decuplat de server, deci aplicația client care va utiliza bean-ul creat anterior va fi executată sub propriul JVM. Așadar, clientul trebuie să aibă acces (cumva) la interfața *bean*-ului, pentru a putea apela metodele din acesta, de la distanță.

O soluție în acest sens ar fi să împachetați sub formă de librărie JAR (**J**ava **A**rchive) codul obiect rezultat după compilarea fișierului sursă ce conține interfața *bean*-ului. Acest fișier nu conține implementarea, deci are sens să fie distribuit și clientului spre utilizare (de aici decuplarea).

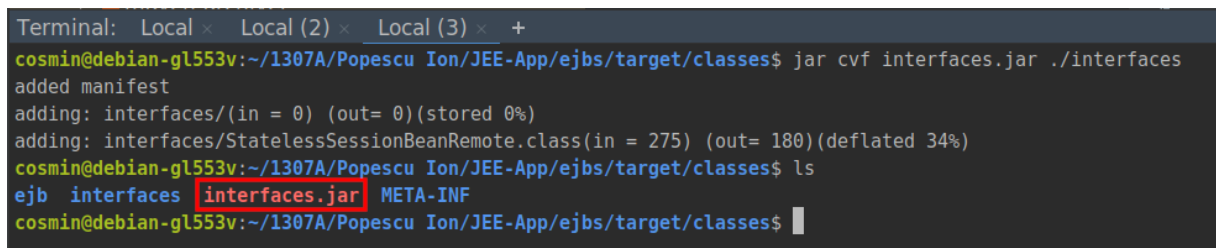
Deschideți un terminal în folder-ul **ejbs/target/classes**. Din IntelliJ, se poate face cu click dreapta pe acest folder în structura de proiect din partea stângă → **Open in Terminal**. Se va deschide un terminal în partea de jos a ferestrei IntelliJ.



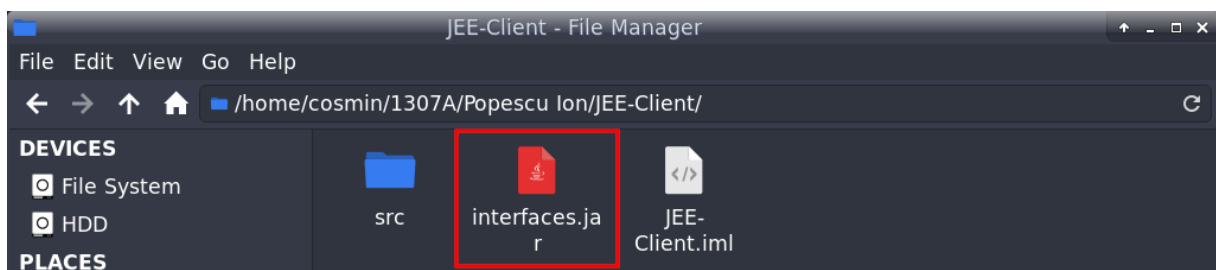
Executați următoarea comandă:

```
jar cvf interfaces.jar ./interfaces
```

Comanda va împacheta conținutul folder-ului **interfaces** într-un fișier JAR.

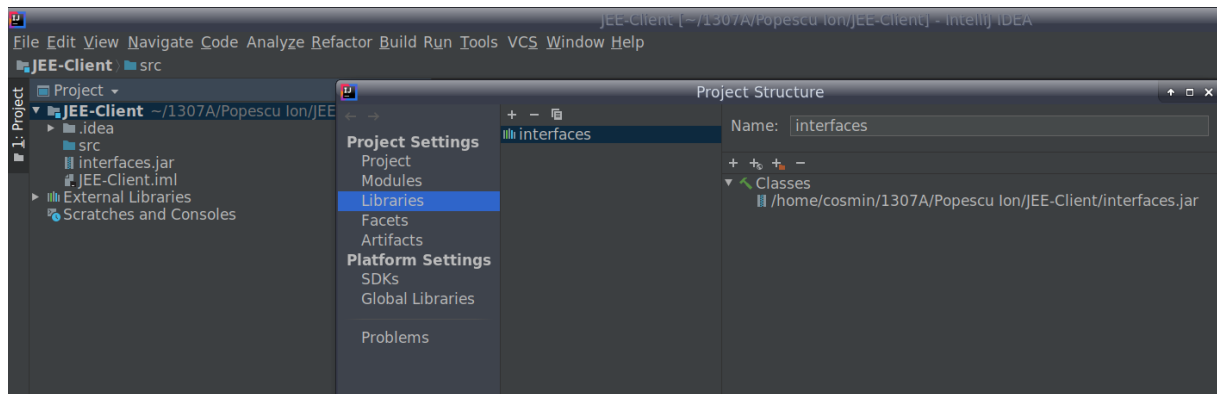


Creați un proiect nou IntelliJ de tip Java (simplu, fără manager de proiect Maven, etc.). Denumiți-l, spre exemplu, **JEE-Client**. Copiați fișierul **interfaces.jar** în folder-ul rădăcină al proiectului.



Acum, trebuie să adăugați acest fișier ca librărie de care proiectul **JEE-Client** depinde (ca aplicația client să poată utiliza interfața **StatelessSessionBeanRemote**). Accesați: **File** → **Project Structure...** → **Libraries** → click pe pictograma în formă de plus (**New Project**

Library) → **Java** → căutați și selectați fișierul **interfaces.jar** copiat în pasul anterior → în fereastra „**Choose modules**” apăsați doar **OK** → apăsați **OK** în fereastra cu setările proiectului.



Creați o clasă Java în folder-ul **src** cu surse, denumită **JEEClient**, având următorul conținut:

```
import interfaces.StatelessSessionBeanRemote;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class JEEClient {
    public static void main(String[] args) throws NamingException {
        Context ctx = new InitialContext();
        StatelessSessionBeanRemote ssb = (StatelessSessionBeanRemote)
ctx.lookup("ssb-example#interfaces.StatelessSessionBeanRemote");
        System.out.println(ssb.getCurrentTime());
        System.out.println("1 + 3 = " + ssb.addNumbers(1, 3));
    }
}
```

În codul de mai sus, se importă interfața *bean*-ului de tip *stateless session*. Apoi, se inițializează un context inițial pentru operația de căutare în JNDI ce urmează (este necesară această inițializare, deoarece toate operațiile de căutare în JNDI sunt relative la un context).

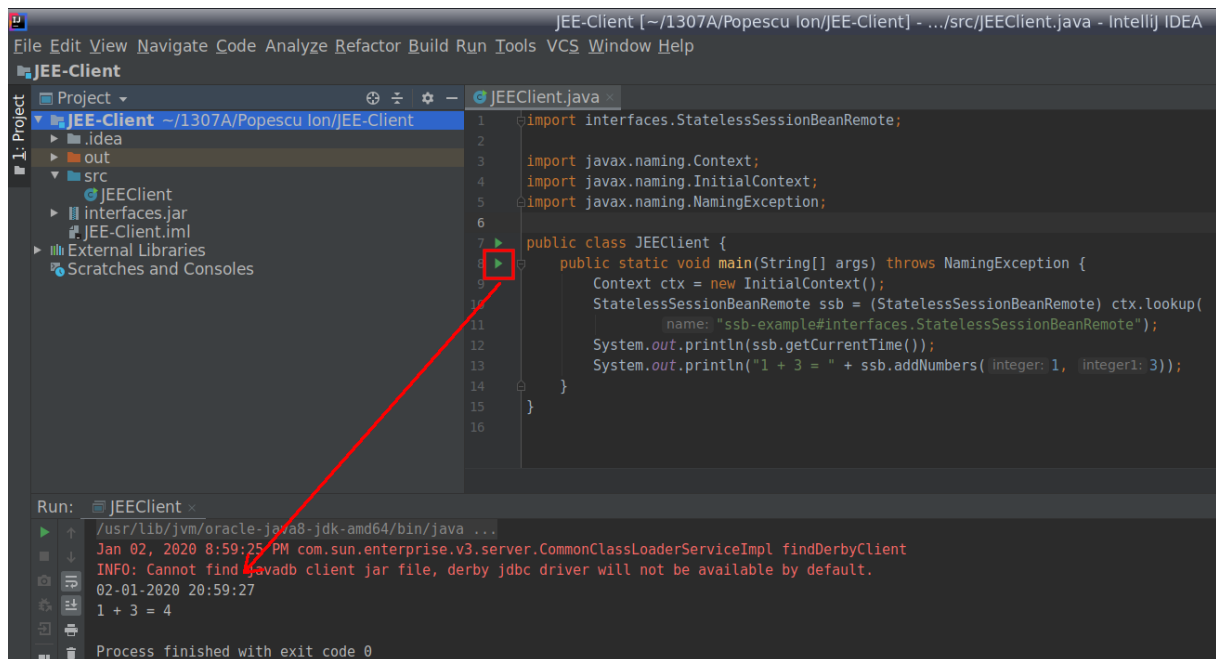
Apoi, se caută în JNDI *bean*-ul expus sub numele pe care l-ați găsit și notat anterior. După ce s-a „pus mâna” pe obiectul stub care apelează metodele remote ale *bean*-ului, se fac apelurile efective de metode: se preia data și ora curentă, respectiv se face calculul complicat care adună numerele 1 și 3.

Totuși, aplicația client nu este pregătită de utilizare, deoarece nu sunt disponibile implementările claselor client specifice GlassFish. Acestea se regăsesc în fișierul **gf-client.jar** din folder-ul **glassfish/lib** al server-ului GlassFish (în acest caz, locația este: **/home/student/opt/glassfish5/glassfish/lib/gf-client.jar**).

Adăugați acest fișier JAR ca dependență la proiect, exact cum ați adăugat și pachetul **interfaces.jar**.

1.12. Testare stateless session bean

Executați aplicația client prin apăsarea butonului verde din dreptul funcției **main()**. IntelliJ va crea automat o configurație de execuție Java.



După apel, dacă consultați log-ul server-ului Glassfish, puteți observa mesajele care au fost generate din conținutul *bean*-ului, după apelarea metodelor încapsulate de acesta:

```

10004 [2020-01-02T18:41:44.755+0200] [glassfish 5.0] [INFO] [] [tid: _ThreadID=107 _ThreadName=Thread-8] [timeMillis: 1577983304755] [
levelValue: 800] [[
10005 [Glassfish] S-a instanțiat un stateless session bean: ejb.StatelessSessionBeanImpl]]
10006
10007 [2020-01-02T18:41:44.755+0200] [glassfish 5.0] [INFO] [] [tid: _ThreadID=107 _ThreadName=Thread-8] [timeMillis: 1577983304755] [
levelValue: 800] [[
10008 [Glassfish] S-a apelat metoda getCurrentTime()]]
10009
10010 [2020-01-02T18:41:44.763+0200] [glassfish 5.0] [INFO] [] [tid: _ThreadID=107 _ThreadName=Thread-8] [timeMillis: 1577983304763] [
levelValue: 800] [[
10011 [Glassfish] S-a apelat metoda addNumbers(1, 3)]]
10012
10013
10014

```

Pentru a observa în timp real apelurile efectuate pe server, deschideți un terminal și executați următoarea comandă:

```
tail -f <LOCAȚIE_GLASSFISH>/glassfish/domains/domain1/logs/server.log
```

Comanda va afișa în timp real ultimele linii din fișierul log, pe măsură ce acesta este populat. Puteți deschide unul sau mai mulți clienți deodată și observați cum *bean*-ul creat este (re)utilizat pentru clienți diferiți.

Atenție: chiar dacă bean-ul creat este de tip *Stateless Session*, nu înseamnă că server-ul va utiliza o singură instanță a acestuia mereu! Presupunând că în *container*-ul de *business* există, pentru moment, o singură instanță a *bean*-ului, iar un client încă „ocupă” acest *bean* (deoarece o metodă apelată încă se execută), atunci server-ul va crea o nouă instanță a *bean*-ului respectiv și o va oferi spre utilizare unui alt client. De aici ideea de „Stateless Session”: nu contează care din *bean*-uri este folosit de care client, deoarece nu există ideea de stare menținută cu un anumit client.

Stateful Session Beans

Spre deosebire de *stateless session beans*, aceste tipuri de *beans* mențin o legătură între clientul apelant și *bean*-ul care a fost apelat. Așadar, dacă n clienți accesează un *stateful session bean*, server-ul este obligat să instanțieze pentru fiecare un *bean* separat, deoarece fiecare *bean* are starea proprie, și nu este permis ca un client să acceseze informații din starea unui *bean* ce

aparține de alt client.

De exemplu, presupunând că pe server există un *bean* ce expune o metodă care afișează lista de e-mail-uri a unui utilizator, cum ar fi ca un alt client să apeleze această metodă din *bean* și să primească lista dvs. de mail-uri, în loc de lista proprie?

Pentru a ilustra funcționarea unui *stateful session bean*, veți crea o aplicație de gestiune a unui cont bancar „deschis” în momentul în care un client accesează prima dată una din operațiunile disponibile:

- depunere numerar
- retragere numerar
- interogare sold

1.13. Codul de business

Mai întâi, creați interfața de *business* a *bean*-ului, care expune unui client operațiunile ce pot fi efectuate. Adăugați un fișier de tip interfață Java în **ejbs/src/main/java/interfaces** și denumiți-l **BankAccountBeanRemote**. Adăugați următorul conținut:

```
package interfaces;

public interface BankAccountBeanRemote {
    Boolean withdraw(Integer amount);
    void deposit(Integer amount);
    Integer getBalance();
}
```

Apoi, creați implementarea corespunzătoare *bean*-ului în pachetul **ejb** din același modul (**ejbs/src/main/java/**). Denumiți clasa Java **BankAccountBeanImpl**, spre exemplu.

```
package ejb;

import interfaces.BankAccountBeanRemote;

import java.io.Serializable;

public class BankAccountBeanImpl implements BankAccountBeanRemote,
    Serializable {
    private Integer availableAmount = 0;

    public Boolean withdraw(Integer amount) {
        if(availableAmount >= amount) {
            availableAmount -= amount;
            return true;
        } else {
            return false;
        }
    }

    public void deposit(Integer amount) {
        availableAmount += amount;
    }

    public Integer getBalance() {
        return availableAmount;
    }
}
```



```
}

```

Înregistrați clasa creată ca și *stateful session bean* în descriptorii XML din fișierul **ejb-jar.xml**, din **ejbs/src/main/resources/META-INF**. Adăugați acest descriptor ca și subordonat al tag-ului **<enterprise-beans>**:

```
<session>
  <ejb-name>BankAccountBean</ejb-name>
  <mapped-name>bankaccount</mapped-name>
  <business-local>interfaces.BankAccountBeanRemote</business-local>
  <business-remote>interfaces.BankAccountBeanRemote</business-remote>
  <ejb-class>ejb.BankAccountBeanImpl</ejb-class>
  <session-type>Stateful</session-type>
</session>
```

De această dată, veți accesa *bean*-ul din *web tier*, spre deosebire de capitoul anterior, unde ați creat o aplicație client Java separată.

1.14. Codul din web tier

În continuare, creați meniul de acces pentru client, ca și pagină JSP. Puteți folosi pagina **index.jsp** deja existentă ca și meniu.

Modificați **servlets/servlet/src/main/webapp/index.jsp** astfel:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<html>
  <head>
    <title>Meniu principal</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h1>Meniu principal</h1>
    <h3>Gestiune cont bancar</h3>
    <form action="./process-bank-operation" method="post">
      <fieldset label="operatiuni">
        <legend>Alegeti operatiunea dorita:</legend>
        <select name="operation">
          <option value="deposit">Depunere numerar</option>
          <option value="withdraw">Retragere
numerar</option>
          <option value="balance">Interogare sold</option>
        </select>
        <br />
        <br />
        Introduceti suma: <input type="number" name="amount"
/>
        <br />
        <br />
        <button type="submit">Efectuare</button>
      </fieldset>
    </form>
  </body>
</html>
```

Pagina conține un formular simplu în care utilizatorul poate selecta operațiunea dorită și

poate introduce o sumă de bani într-un câmp numeric, utilizat la operațiunile de retragere / depunere.

Ținta formularului este un servlet mapat pe ruta de acces **./process-bank-operation**, către care datele din formular sunt trimise prin metoda **POST**. Așadar, acum veți crea servlet-ul respectiv: o clasă Java denumită **ProcessBankOperationServlet**, plasată în modulul **servlet**, în următorul folder: **servlets/servlet/src/main/java**. Adăugați codul următor:

```
import interfaces.BankAccountBeanRemote;

import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class ProcessBankOperationServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        // preluare parametri din cererea HTTP
        String operation = request.getParameter("operation");
        String amountString = request.getParameter("amount");

        // nu conteaza suma introdusa in campul numeric daca
        operatiunea este de tip "Sold cont"
        Integer amount = (!amountString.equals("")) ?
        Integer.parseInt(amountString) : 0;

        // se incearca preluarea bean-ului folosind obiectul
        HttpSession, care pastreaza o sesiune HTTP intre client si server
        BankAccountBeanRemote bankAccount;
        bankAccount =
        (BankAccountBeanRemote) request.getSession().getAttribute("bankAccount-
        Bean");

        // daca nu exista nimic pastrat in sesiunea HTTP, inseamna ca
        bean-ul se preia prin JNDI lookup
        if (bankAccount == null) {
            try {
                InitialContext ctx = new InitialContext();
                bankAccount = (BankAccountBeanRemote)
                ctx.lookup("bankaccount#interfaces.BankAccountBeanRemote");

                // dupa preluarea bean-ului prin JNDI, obiectul se
                stocheaza in sesiune pentru a fi refolosit ulterior
                // cererile urmatoare vor utiliza obiectul remote
                stocat in sesiune
                request.getSession().setAttribute("bankAccountBean",
                bankAccount);
            } catch (NamingException e) {
                e.printStackTrace();
                return;
            }
        }
    }
}
```

```

    }
}

Integer accountBalance = null;
String message = "";

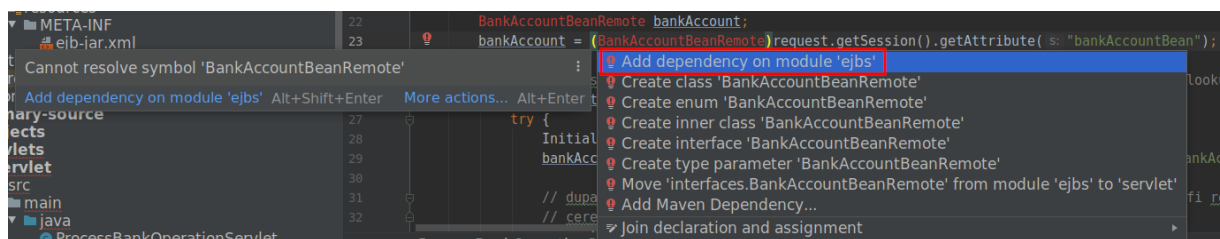
// in functie de operatia selectata de client, se apeleaza
metoda corespunzatoare din obiectul remote
if (operation.equals("deposit")) {
    bankAccount.deposit(amount);
    message = "In contul dvs. au fost depusa suma: " + amount
+ ".";
} else if (operation.equals("withdraw")) {
    if (bankAccount.withdraw(amount)) {
        message = "Din contul dvs. s-a retras suma de: " +
amount + ".";
    } else {
        message = "Operatiunea a esuat! Fonduri
insuficiente.";
    }
} else if (operation.equals("balance")) {
    accountBalance = bankAccount.getBalance();
}

message += "<br /><br />";
if (accountBalance != null) {
    message += "Sold cont: " + accountBalance;
}
message += "<br /><a href='./'>Inapoi la meniul
principal</a>";

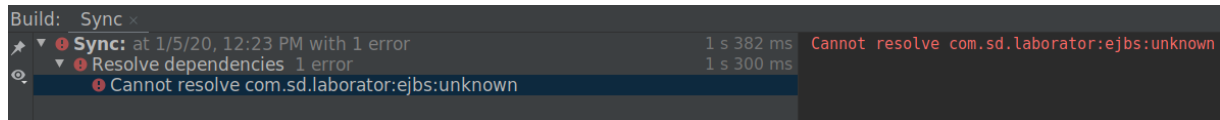
// dupa construirea mesajului raspuns, acesta este trimis ca
si continut HTML inapoi la clientul apelant
response.setContentType("text/html");
response.getWriter().print(message);
}
}

```

După ce scrieți codul respectiv în fișierul sursă, veți observa o problemă: nu aveți acces din acest modul la pachetul cu *bean-uri enterprise (ejbs)*, și deci nu se poate importa interfața **BankAccountBeanRemote**. Pentru a rezolva acest lucru, modulul Maven **ejbs** trebuie adăugat ca dependență la modulul Maven **servlet**. IntelliJ ajută utilizatorul în acest sens: dați click pe **BankAccountBeanRemote** marcat cu roșu, apăsați **ALT+ENTER** și alegeți „Add dependency on module ejbs”.



Nu este suficient, deoarece IntelliJ nu poate detecta singur ce tip de modul este cel importat.



Așa încât, modificați manual fișierul `servlets/servlet/pom.xml`, iar pentru dependența adăugată de IntelliJ, specificați și atributul `<type>ejb</type>` (deoarece se importă un modul de tip *Enterprise Java Bean*, conform schemei DTD al fișierului POM).

```
<dependency>
  <groupId>com.sd.laborator</groupId>
  <artifactId>ejbs</artifactId>
  <type>ejb</type>
</dependency>
```

Nu uitați de „**Import changes**” pentru a sincroniza proiectul Maven și modulele sale.

Corpul servlet-ului conține cod de tratare a cererilor de tip POST, deoarece formularul care îl țintește trimite datele prin această metodă HTTP (se putea folosi și GET, spre exemplu).

Bean-ul de tip *stateful* se preia prin **JNDI lookup** prima dată când clientul accesează servlet-ul. În acel moment, server-ul îi creează și asociază o instanță a *bean*-ului în contextul de execuție, iar această instanță este reținută de servlet în sesiunea HTTP, deoarece este nevoie de ea la fiecare cerere. **Dacă nu ați fi reținut instanța primită, cererile ulterioare vor prelua instanțe noi de bean de la server (chiar dacă este stateful! Starea este păstrată pe aceeași instanță. În momentul în care se face lookup din nou, e ca și cum ați cere o instanță nouă, deci ați pierdut starea).**

Cererile ulterioare primei vor prelua *bean*-ul din sesiune, iar apelurile metodelor acestuia vor ține cont de starea anterioară, deoarece clientul este același, iar server-ul *enterprise* va folosi aceeași instanță din memorie pentru a deservi clientul respectiv. Dacă un alt client cere o instanță de *bean*, server-ul va crea o instanță nouă și i-o va asocia lui, păstrându-i starea în cererile ulterioare, ș.a.m.d, pentru fiecare client **diferit**.

Nu uitați să creați descriptorul XML pentru servlet, pentru ca server-ul *enterprise* să îl mapeze pe ruta dorită și să îl gestioneze corespunzător.

Deschideți `servlets/servlet/src/main/webapp/WEB-INF/web.xml` și adăugați o nouă mapare pentru servlet-ul **ProcessBankOperationServlet**:

```
<servlet>
  <servlet-name>ProcessBankOperation</servlet-name>
  <servlet-class>ProcessBankOperationServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ProcessBankOperation</servlet-name>
  <url-pattern>/process-bank-operation</url-pattern>
</servlet-mapping>
```

Urmează pașii cu care deja v-ați obișnuit: **compile** → **package** → **redeploy de la consolă**.

1.15. Testare stateful session bean

Testarea *bean*-ului de tip *stateful* presupune existența a 2 clienți diferiți (sesiuni diferite HTTP). Aveți 2 variante, întrucât server-ul este local: folosiți 2 browsere diferite, sau folosiți 1 singur browser, o dată din modul obișnuit de navigare, și a doua oară din modul **incognito**.

Deschideți un browser și navigați la adresa:

<http://localhost:8080/servlet/>



Meniu principal

Gestiune cont bancar

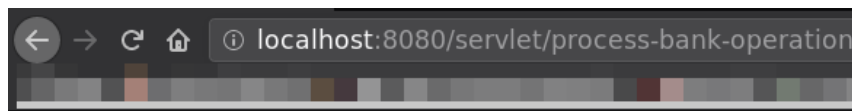
Alegeti operatiunea dorita:

Depunere numerar ▾

Introduceti suma:

Efectuare

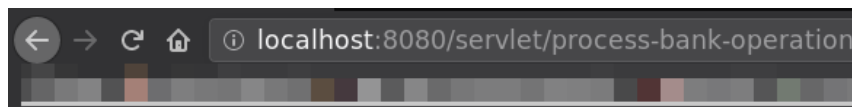
Selecțai **Depunere numerar**, introduceți o sumă și apoi apăsați pe „**Efectuare**”. În acest moment, clientului curent (browser-ul web, în acest caz) a primit o instanță de *stateful session bean*, iar aceasta a fost stocată în sesiune.



In contul dvs. au fost depusa suma: 120.

[Inapoi la meniul principal](#)

Înapoi la meniul principal și verificați soldul. Valoarea este preluată din *bean*-ul existent în memoria server-ului *enterprise*.



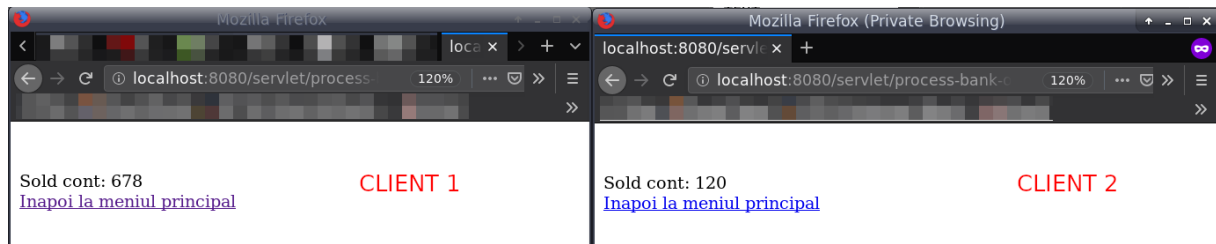
Sold cont: 120

[Inapoi la meniul principal](#)

Puteți testa alte operațiuni de retragere / depunere pentru a confirma că starea *bean*-ului este păstrată de la un apel la altul.

Acum deschideți un al 2-lea browser (sau modul incognito dacă folosiți același browser: **CTRL+SHIFT+N** pentru Google Chrome sau **CTRL+SHIFT+P** pentru Mozilla Firefox).

Faceți o depunere și observați că soldul diferă de cel al primului client, deoarece acesta este un client nou, deci server-ul va utiliza o altă instanță de *bean* atunci când se face căutarea (*lookup*) în JNDI.



JPA Entities

Entitățile JPA (*JPA Entities*) au fost precedate, ca tehnologie, de *Entity Beans*. Începând de la standardul EJB 3.0, *Entity Bean*-urile sunt marcate ca învechite și, de aceea, veți folosi tehnologia nouă în acest capitol al laboratorului.

1.16. Configurare persistență

Pentru a folosi *Java Persistence API* în proiectul JEE, trebuie să adăugați un descriptor de persistență într-un fișier denumit **persistence.xml**, ce rezidă în folder-ul **ejbs/src/main/resources/META-INF**. Creați fișierul și adăugați următorul conținut:

```
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
version="2.0"
xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="bazaDeDateSQLite" transaction-
type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>ejb.StudentEntity</class>
    <properties>
      <property name="javax.persistence.jdbc.driver"
value="org.sqlite.JDBC" />
      <property name="javax.persistence.jdbc.url"
value="jdbc:sqlite:/home/student/studenti.db" />
      <property name="eclipselink.logging.level" value="ALL" />
      <property name="eclipselink.ddl-generation" value="create-
tables" />
    </properties>
  </persistence-unit>
</persistence>
```

În acest XML s-a definit o așa-numită unitate de persistență (*persistence unit*), adică o grupare logică a tuturor entităților gestionate de instanțele unui **EntityManager** (detalii despre această clasă în cele ce urmează) în aplicația JEE. Unitatea de persistență poate folosi o bază de date SQL / NoSQL, o bază de date *embedded* etc. În acest caz s-a utilizat o bază de date SQLite, în care tranzațiile se fac doar cu „resurse locale”, deoarece această bază de date este complet încapsulată într-un fișier cu extensia **.db**. S-a marcat acest lucru prin atributul **transaction-type="RESOURCE_LOCAL"**.

JPA trebuie configurat în așa fel încât să știe ce furnizor și *driver* de persistență să folosească atunci când utilizatorul lucrează cu acest API și cu datele ce se transmit. În acest caz,

furnizorul (*provider*-ul) de persistență este interfața **PersistenceProvider** din implementarea JPA **EclipseLink** - disponibilă implicit odată cu server-ul GlassFish.

Pentru unitatea de persistență se specifică, la început (după tag-ul **<provider>**), care sunt clasele ce vor fi gestionate de **EntityManager** - clasele de domeniu, adică cele care sunt mapate în baza de date sub formă de tabele.

Apoi, se dau proprietățile furnizorului de persistență: driver-ul folosit, calea pe disc către fișierul unde este încapsulată baza de date SQLite (în acest caz, **/home/student/studenti.db**), nivelul de *logging* și, la final, este specificat faptul că EclipseLink va crea schema bazei de date folosind o interogare de tip „**CREATE TABLE...**” în momentul în care aplicația este încărcată pe server.

Atenție: fișierul pe care îl specificați ca și bază de date SQLite trebuie să se afle într-o locație în care utilizatorul care a pornit server-ul GlassFish are drepturi de scriere!

Pentru ca aplicația JEE să poată utiliza *driver*-ul JDBC pentru SQLite (clasa **org.sqlite.JDBC**), acest driver trebuie adăugat ca dependență în modulul Maven în care este folosit, adică în modulul **ejbs**. Deschideți fișierul **ejbs/pom.xml** și adăugați următoarea dependență:

```
<dependency>
  <groupId>org.xerial</groupId>
  <artifactId>sqlite-jdbc</artifactId>
  <version>3.30.1</version>
</dependency>
```

1.17. Adăugare entitate JPA

Creați o entitate JPA denumită **StudentEntity**, ca și clasă Java situată în **/ejbs/src/main/java/ejb**. Codul clasei este următorul:

```
package ejb;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class StudentEntity {
    @Id
    @GeneratedValue
    private int id;
    private String nume;
    private String prenume;
    private int varsta;

    public StudentEntity() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

```

    public String getNume() {
        return nume;
    }

    public void setNume(String nume) {
        this.nume = nume;
    }

    public String getPrenume() {
        return prenume;
    }

    public void setPrenume(String prenume) {
        this.prenume = prenume;
    }

    public int getVarsta() {
        return varsta;
    }

    public void setVarsta(int varsta) {
        this.varsta = varsta;
    }
}

```

Observați că entitatea JPA seamănă cu un **POJO** (*Plain Old Java Object*) ce conține adnotări corespunzătoare, unde este cazul: adnotarea **@Entity** marchează faptul că această clasă este o entitate JPA, adnotarea **@Id** indică după care câmp vor fi identificate înregistrările din tabela rezultată în baza de date (cheia primară), adnotarea **@GeneratedValue** marchează faptul că acel câmp adnotat (în acest caz, câmpul **id**) conține o valoare generată automat la inserarea unei înregistrări - nu trebuie dat explicit de utilizator.

Câmpurile încapsulate trebuie să fie tipuri de date simple (primitive sau nu), și să fie serializabile.

1.18. Codul din web tier

Pentru a ilustra lucrul cu această entitate JPA simplă, veți folosi același formular de introducere a datelor unui student, utilizat în laboratorul 1. Adăugați codul acestuia în fișierul **servlets/servlet/src/main/webapp/formular.jsp**.

```

<html xmlns:jsp="http://java.sun.com/JSP/Page">
  <head>
    <title>Formular student</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <h3>Formular student</h3>
    Introduceti datele despre student:
    <form action="./process-student" method="post">
      Nume: <input type="text" name="nume" />
      <br />
      Prenume: <input type="text" name="prenume" />
      <br />
      Varsta: <input type="number" name="varsta" />
      <br />
    </form>
  </body>
</html>

```



```

        <br />
        <button type="submit" name="submit">Trimite</button>
    </form>
</body>
</html>

```

Formularul va prelua aceste date și le va trimite unui servlet spre procesare. Servlet-ul preia datele din cererea HTTP și le persistă într-o bază de date *embedded* de tip **SQLite** (încapsulată într-un fișier pe disc). Creați acest servlet în **servlets/servlet/src/main/java/ProcessStudentServlet.java**, cu următorul conținut:

```

import ejb.StudentEntity;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class ProcessStudentServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        // se citesc parametrii din cererea de tip POST
        String nume = request.getParameter("nume");
        String prenume = request.getParameter("prenume");
        int varsta = Integer.parseInt(request.getParameter("varsta"));

        // pregatire EntityManager
        EntityManagerFactory factory =
        Persistence.createEntityManagerFactory("bazaDeDateSQLite");
        EntityManager em = factory.createEntityManager();

        // creare entitate JPA si populare cu datele primite din
        formular
        StudentEntity student = new StudentEntity();
        student.setNume(nume);
        student.setPrenume(prenume);
        student.setVarsta(varsta);

        // adaugare entitate in baza de date (operatiune de
        persistenta)
        // se face intr-o tranzactie
        EntityTransaction transaction = em.getTransaction();
        transaction.begin();
        em.persist(student);
        transaction.commit();

        // inchidere EntityManager
        em.close();
        factory.close();
    }
}

```

```

        // trimitere raspuns inapoi la client
        response.setContentType("text/html");
        response.getWriter().println("Datele au fost adaugate in baza
de date." +
        "<br /><br /><a href='./'>Inapoi la meniul
principal</a>");
    }
}

```

Clasa **EntityManager** este folosită, după cum este și denumită, la gestiunea tuturor entităților persistente (*JPA Entities*) de care este nevoie în aplicația JEE. O instanță **EntityManager** se obține de la **EntityManagerFactory**, pe baza numelui unității de persistență.

Creați încă un servlet folosit pentru afișarea listei de studenți disponibilă în baza de date. Denumiți servlet-ul **FetchStudentListServlet** și adăugați următorul conținut în fișierul sursă:

```

import ejb.StudentEntity;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;

public class FetchStudentListServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        // pregatire EntityManager
        EntityManagerFactory factory =
        Persistence.createEntityManagerFactory("bazaDeDateSQLite");
        EntityManager em = factory.createEntityManager();

        StringBuilder responseText = new StringBuilder();
        responseText.append("<h2>Lista studenti</h2>");
        responseText.append("<table
border='1'><thead><tr><th>ID</th><th>Nume</th><th>Prenume</th><th>Vars
ta</th></thead>");
        responseText.append("<tbody>");

        // preluare date studenti din baza de date
        TypedQuery<StudentEntity> query = em.createQuery("select
student from StudentEntity student", StudentEntity.class);
        List<StudentEntity> results = query.getResultList();
        for (StudentEntity student : results) {
            // se creeaza cate un rand de tabel HTML pentru fiecare
student gasit
            responseText.append("<tr><td>" + student.getId() +

```

```

"</td><td>" +
                student.getNume() + "</td><td>" +
student.getPrenume() +
                "</td><td>" + student.getVarsta() + "</td></tr>");
    }

    responseText.append("</tbody></table><br /><br /><a
href='.'/'>Inapoi la meniul principal</a>");

    // inchidere EntityManager
    em.close();
    factory.close();

    // trimitere raspuns la client
    response.setContentType("text/html");
    response.getWriter().print(responseText.toString());
}
}

```

Observați că preluarea listei de studenți se face printr-o interogare care seamănă cu limbajul SQL. Acest limbaj se numește **JPQL (Java Persistence Query language)**.

Nu uitați de maparea servleților prin descriptori XML. Modificați **web.xml** astfel:

```

...

<servlet>
    ...
</servlet>
<servlet>
    <servlet-name>ProcessStudent</servlet-name>
    <servlet-class>ProcessStudentServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>FetchStudentList</servlet-name>
    <servlet-class>FetchStudentListServlet</servlet-class>
</servlet>

<servlet-mapping>
    ...
</servlet-mapping>
<servlet-mapping>
    <servlet-name>ProcessStudent</servlet-name>
    <url-pattern>/process-student</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>FetchStudentList</servlet-name>
    <url-pattern>/fetch-student-list</url-pattern>
</servlet-mapping>

...

```

Modificați meniul principal din pagina **index.jsp** ca să includeți ancore către formularul de adăugare a unui student, respectiv către servlet-ul care afișează lista studenților:

```

...
...

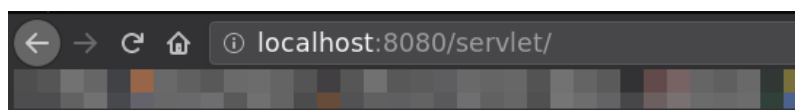
```

```
</form>
<hr />
<h3>Baza de date cu studenti</h3>
<a href="./formular.jsp">Adaugare student</a>
<br />
<a href="./fetch-student-list">Afisare lista studenti</a>
...
...
```

1.19. Testare entitate JPA

Curățați aplicația (**clean**), apoi compilați (**compile**), împachetați aplicația (**package**) în artefact EAR și încărcați-o pe server (**redeploy** dacă ați mai încărcat-o anterior, sau **deploy** dacă e prima dată când încărcați - nu uitați, **de la consolă**).

Navigați la <http://localhost:8080/servlet>.



Meniu principal

Gestiune cont bancar

Alegeti operatiunea dorita: _____

Depunere numerar ▾

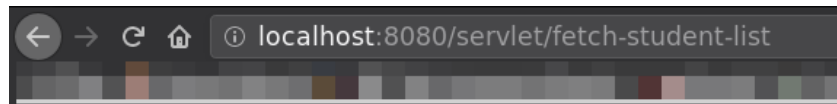
Introduceti suma:

Efectuare

Baza de date cu studenti

[Adaugare student](#)
[Afisare lista studenti](#)

Adăugați studenți în baza de date din meniul corespunzător, apoi testați dacă lista de studenți poate fi preluată din baza de date (din fișierul **studenti.db**).



Lista studenti

ID	Nume	Prenume	Varsta
1	Popescu	Ion	23
51	Ala	Bala	13
101	Porto	Cala	40

[Inapoi la meniul principal](#)

Aplicații și teme

• Temă de laborator

Adăugați operațiile de actualizare și de ștergere a unui student din baza de date SQLite:

- modificați meniul principal din pagina **index.jsp** pentru a indica aceste operații
- adăugați încă 2 servleți corespunzători acestor operații. În acești servleți, folosiți clasa **EntityManager** și limbajul **JPQL** (*Java Persistence Query Language*) pentru a implementa operațiile cerute. Sugestie:

- actualizare: căutați studentul în baza de date după nume și / sau prenume, apoi preluați identificatorul acestuia (ID-ul). Având acest ID, faceți o operație de UPDATE folosind JPQL cu noile date preluate de la utilizator (eventual dintr-un formular JSP).

- ștergere: căutați studentul în baza de date după nume și / sau prenume, apoi preluați identificatorul acestuia (ID-ul). Având acest ID, faceți o operație de DELETE folosind JPQL, după ID-ul preluat.

- **Alternativă:** atunci când se preiau studenții pentru afișare, puteți prelua și ID-ul din baza de date și să îl setați ca parametru URL pentru servlet-ul apelat la click pe un buton de ștergere / actualizare pus în dreptul fiecărui student, asemănător cu:

```
<a href="/update-student?id=ID&nume=NUME&prenume=...">Actualizeaza</a>
<a href="/delete-student?id=ID">Sterge</a>
```

• Temă pe acasă

Adăugați încă o tabelă în baza de date **studenti.db**, numită **Cursuri** (deci o nouă clasă de tip *JPA Entity*). Pentru această nouă entitate, implementați toate operațiile de adăugare, preluare, actualizare, ștergere, punându-le în evidență printr-un formular JSP asemănător cu cel din laborator.

Bibliografie

Enterprise Java Beans - https://docs.oracle.com/cd/E24329_01/web.1211/e24446/ejbs.htm

Accesarea bean-urilor - <https://docs.oracle.com/javaee/7/tutorial/ejb-intro004.htm>

Introducere în Java Persistence API - <https://docs.oracle.com/javaee/7/tutorial/persistence-intro.htm>

JPA Entities - https://docs.oracle.com/cd/E16439_01/doc.1013/e13981/undejbs003.htm

Persistence Units - <https://docs.oracle.com/cd/E19798-01/821-1841/bnbri/index.html>

JPQL Language reference - https://docs.oracle.com/html/E13946_04/ejb3_langref.html

Entity Beans - https://docs.oracle.com/cd/B14099_19/web.1012/b15505/entity.htm

Message-Driven Beans - <https://docs.oracle.com/javaee/6/tutorial/doc/gipko.html>