Using Hibernate with a network DB

Define environment

Make sure you have defined DERBY_HOME

e.g., put in your bash_profile:

export DERBY_HOME="$HOME/Download/db-derby-10.15.2.0-bin"

Basic (network) server operations

```
# start the network server
java -jar $DERBY_HOME/lib/derbyrun.jar server start &

#get info about the server
./NetworkServerControl sysinfo

#shudown server
java -jar $DERBY_HOME/lib/derbyrun.jar server shutdown
```

Interacting with the (EMBEDDED) server using ij

```
java -jar $DERBY_HOME/lib/derbyrun.jar ij
CONNECT 'jdbc:derby:firstdb;create=true';
CREATE TABLE FIRSTTABLE (ID INT PRIMARY KEY, NAME VARCHAR(12));
INSERT INTO FIRSTTABLE VALUES (10,'TEN'),(20,'TWENTY'),(30,'THIRTY');
SELECT * FROM FIRSTTABLE;
DROP TABLE FIRSTTABLE;
exit;
```

https://db.apache.org/derby/docs/10.15/getstart/getstartderby.pdf page 24

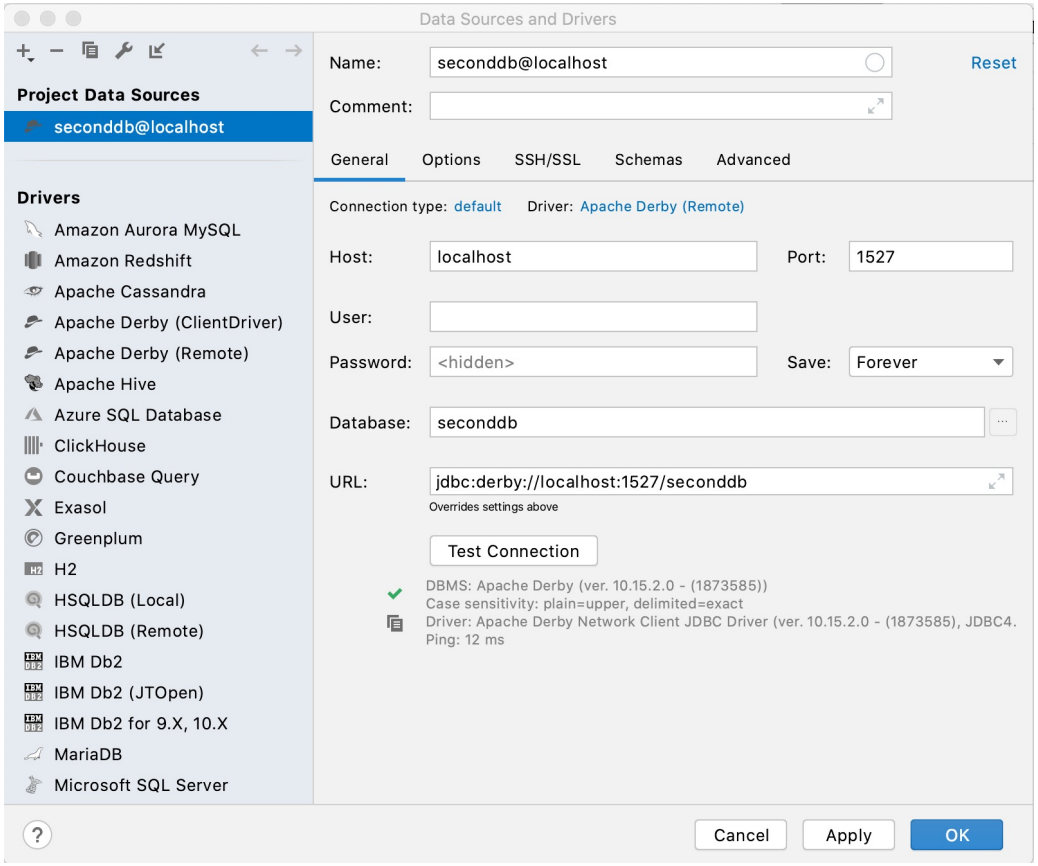
Interacting with the (NETWORK) server using ij

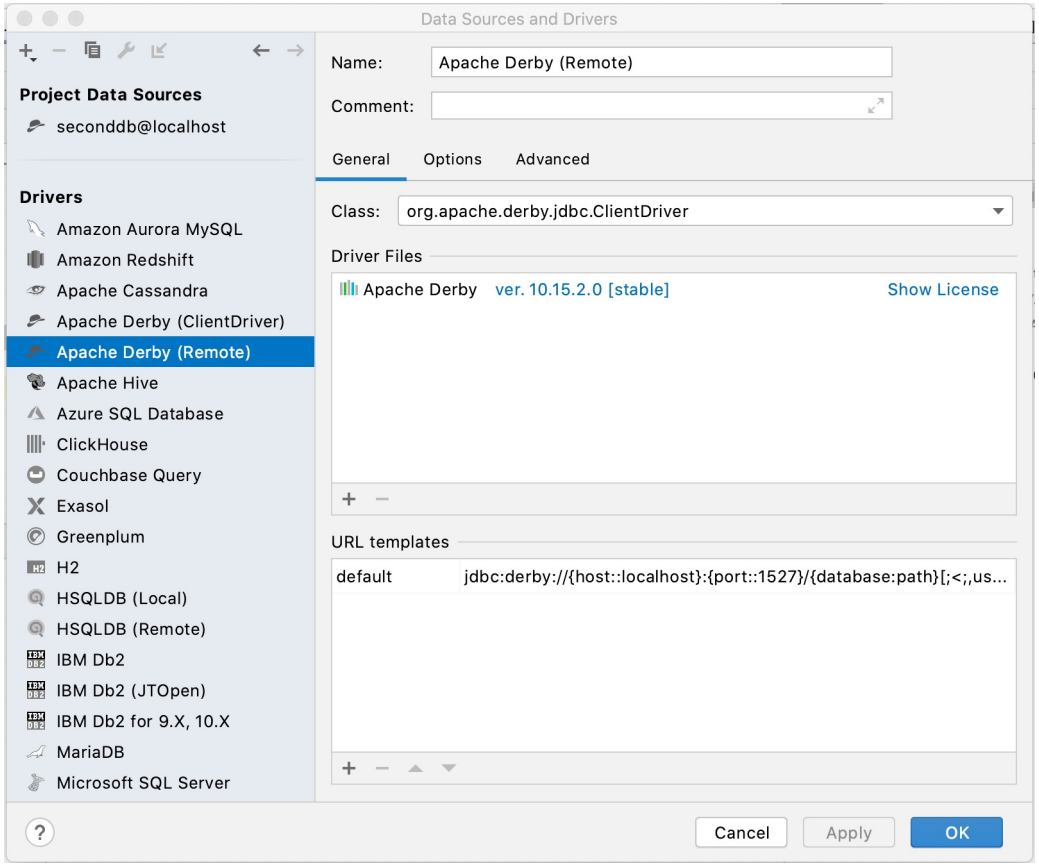
```
java -jar $DERBY_HOME/lib/derbyrun.jar server start &
java -jar $DERBY_HOME/lib/derbyrun.jar ij
CONNECT 'jdbc:derby://localhost:1527/seconddb;create=true';
SHOW CONNECTIONS;
CREATE TABLE EMPLOYEE( ID INTEGER not null GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1) constraint  EMPLOYEE_PK primary key,  FIRSTNAME VARCHAR(30), LASTNAME  VARCHAR(30) );
INSERT INTO EMPLOYEE (FIRSTNAME, LASTNAME) VALUES ('Valentino','Rossi'),('Sofia','Goggia');
SELECT * FROM EMPLOYEE;
exit;
```

https://db.apache.org/derby/docs/10.15/getstart/getstartderby.pdf page 27

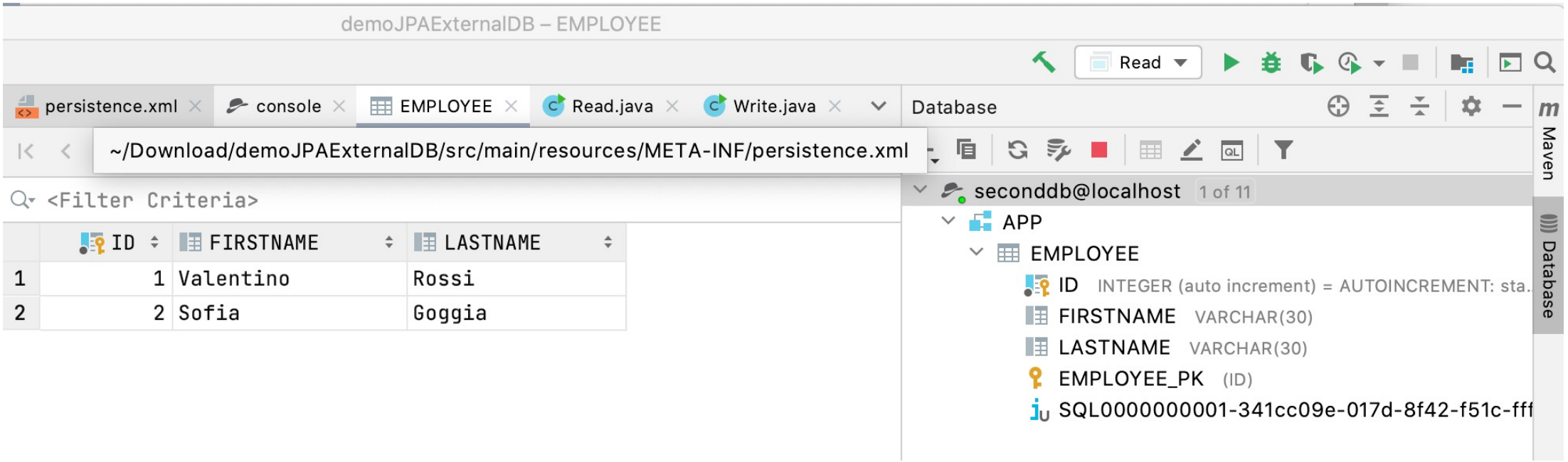
Viewing it from IntelliJ

In a clone of project demoJPA:





Viewing it from IntelliJ



Viewing it from code in IntelliJ

ADD POM DEPENDENCY!

```
<dependency>
   <groupId>org.apache.derby</groupId>
   <artifactId>derbyclient</artifactId>
   <version>10.15.2.0</version>
</dependency>
```

VERY IMPORTANT!

MODIFY PERSISTENCE.XML

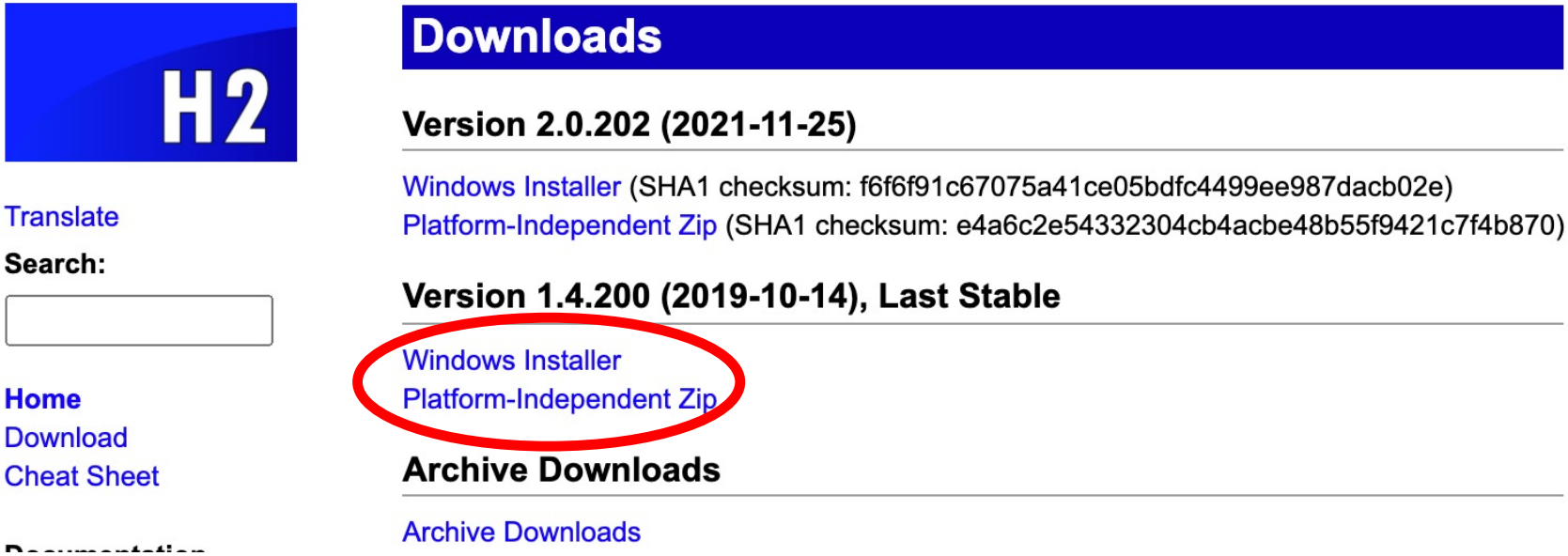
```
<properties>
   <property name="hibernate.connection.url" value="jdbc:derby://localhost:1527/seconddb"/>
   <property name="hibernate.connection.driver_class" value="org.apache.derby.jdbc.ClientDriver"/>
```

Note: since we now have an autoincrementing key in the DB,
we must change the generation strategy in EmployeeEntity.java:
it was: @GeneratedValue(strategy=*SEQUENCE*)
it is now: @GeneratedValue(strategy = GenerationType.*IDENTITY*)

see also https://thorben-janssen.com/jpa-generate-primary-keys/

Changing DBMS: H2

Install H2

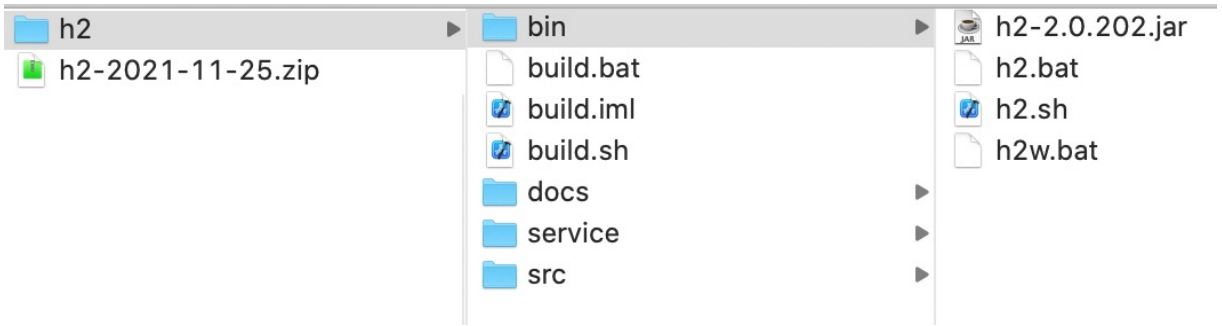


DOWNLOAD:

http://www.h2database.com/html/download.html

TUTORIAL:

https://www.h2database.com/html/tutorial.html



Create new DB with H2

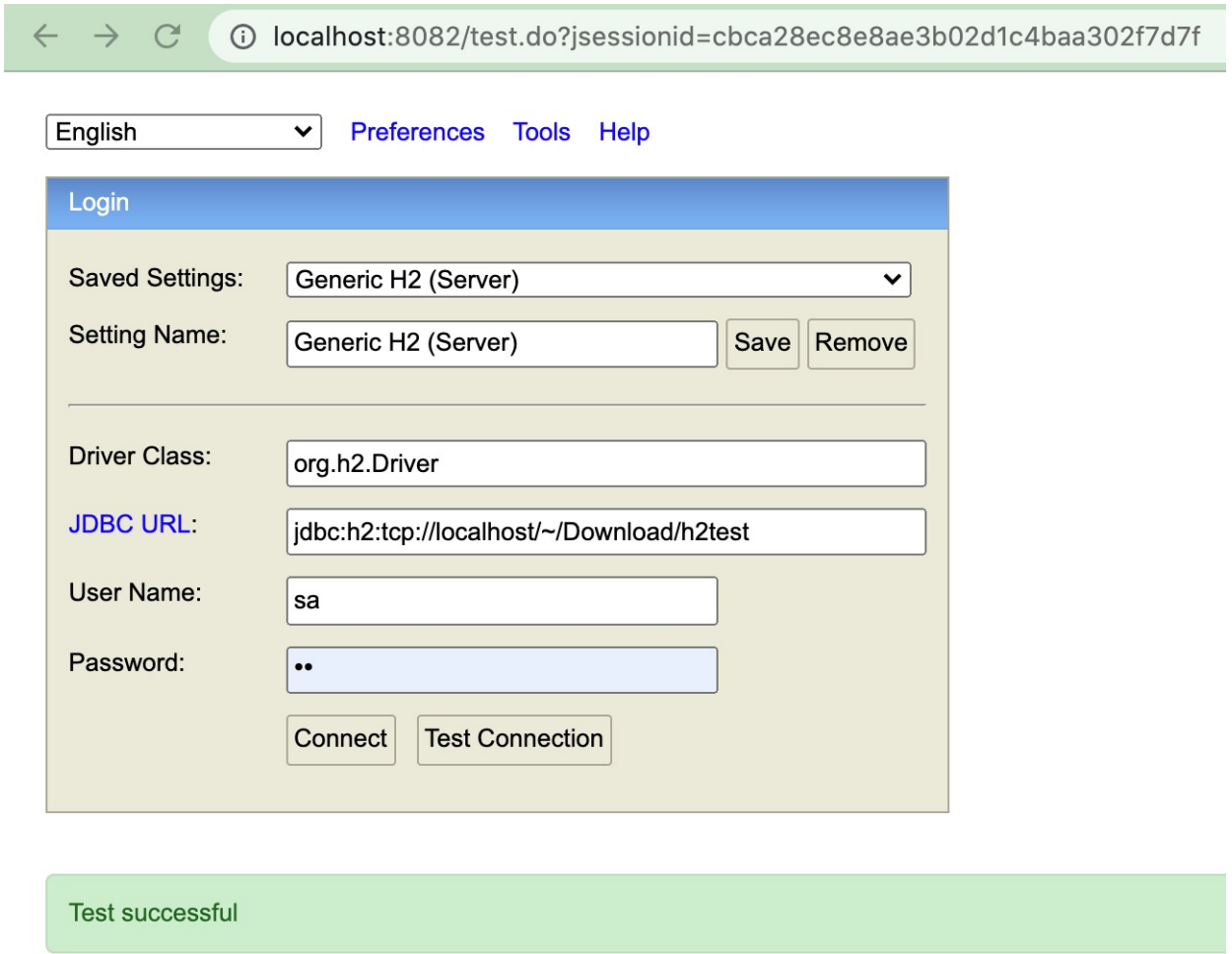
`-cp <class search path of directories and zip/jar files>`

cd into h2/bin

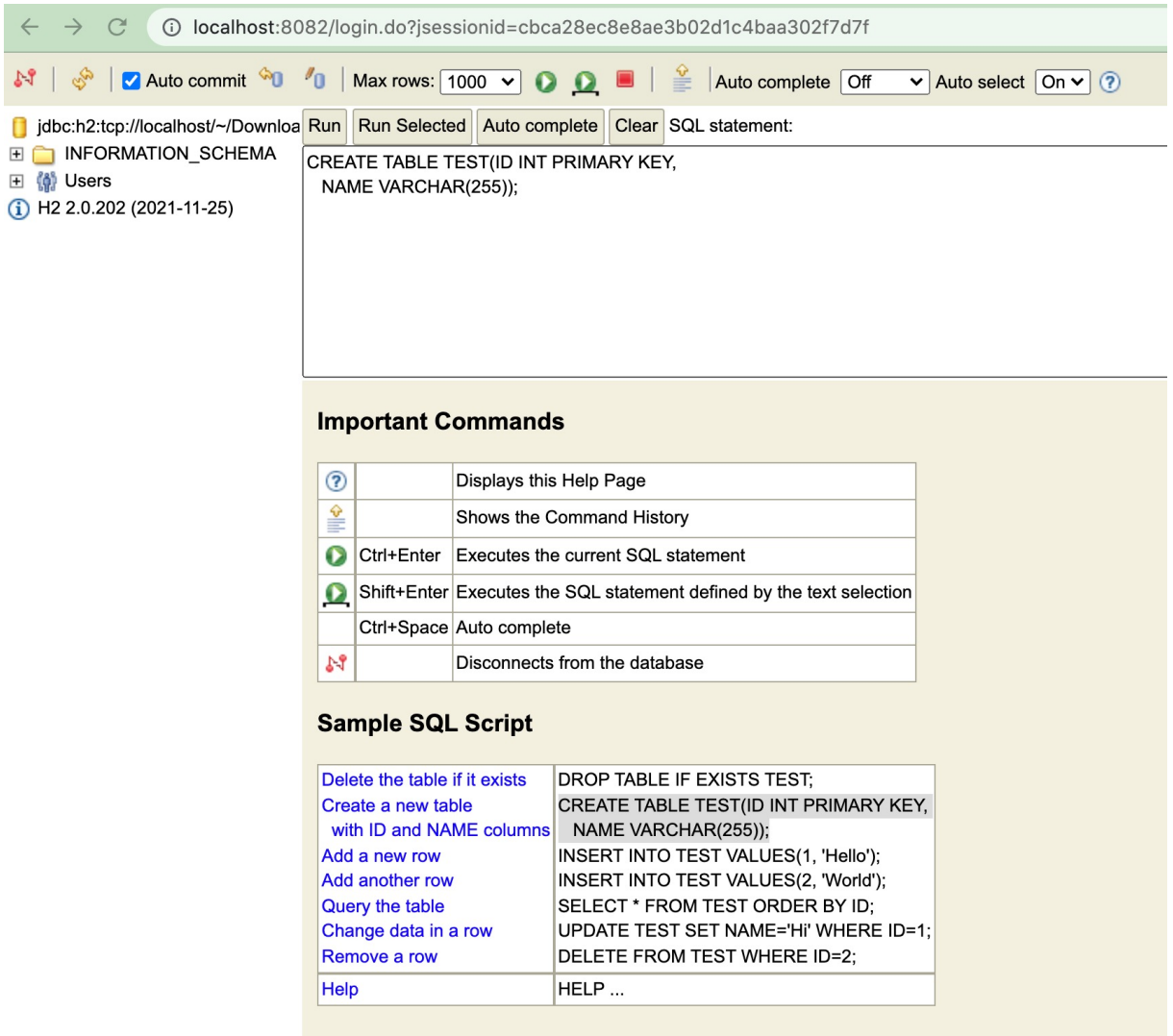
```
MR-MacBookPro:bin ronchet$  java -cp h2-*.jar org.h2.tools.Shell

Welcome to H2 Shell 2.0.202 (2021-11-25)
Exit with Ctrl+C
[Enter]   jdbc:h2:tcp://localhost/~/Download/h2test
URL       jdbc:h2:tcp://localhost/~/Download/h2test
[Enter]   org.h2.Driver
Driver    org.h2.Driver
[Enter]
User      sa
Password
Type the same password again to confirm database creation.
Password
Connected
Commands are case insensitive; SQL statements end with ';'
help or ?      Display this help
list           Toggle result list / stack trace mode
maxwidth       Set maximum column width (default is 100)
autocommit     Enable or disable autocommit
history        Show the last 20 statements
quit or exit   Close the connection and exit

sql> exit
Connection closed
```

Access H2 with browser

java -jar h2*.jar



Access H2 with browser



Access H2 with browser

ALSO:

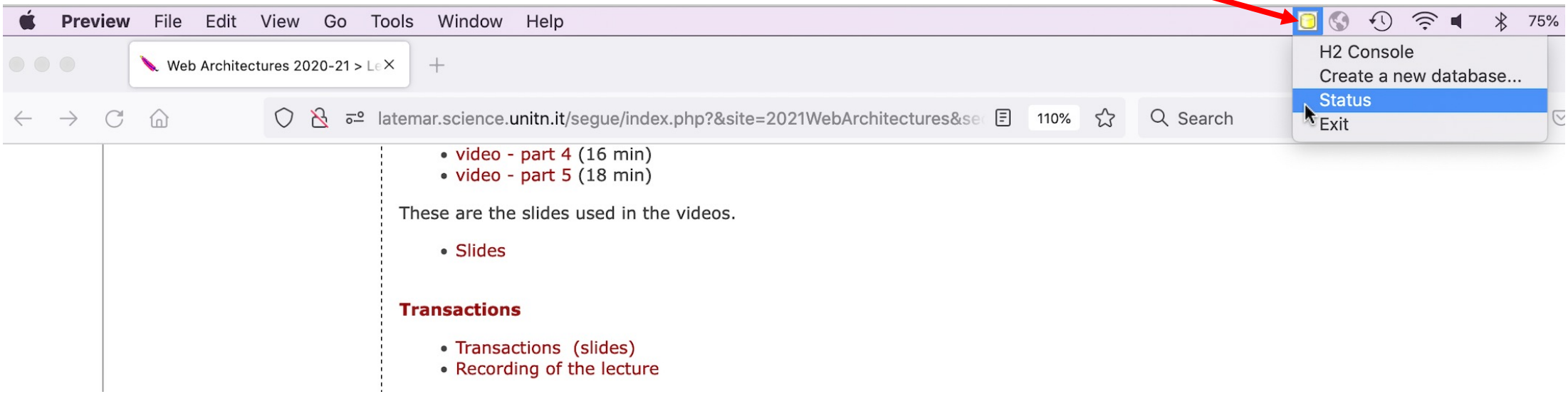


Table definition

```
CREATE TABLE EMPLOYEE( ID INTEGER not null GENERATED ALWAYS AS IDENTITY constraint  EMPLOYEE_PK primary key,  FIRSTNAME VARCHAR(30), LASTNAME  VARCHAR(30) );

INSERT INTO EMPLOYEE (FIRSTNAME, LASTNAME) VALUES ('Valentino','Rossi'),('Sofia','Goggia');

SELECT * FROM EMPLOYEE;

exit;
```

Viewing it from code in IntelliJ

CHANGE POM DEPENDENCY!

```
<dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>2.0.202</version>
 </dependency>
```

VERY IMPORTANT!

MODIFY PERSISTENCE.XML

```
 <class>it.unitn.disi.ronchet.demojpa.entities.EmployeeEntity</class>
       <properties>
         <property name="hibernate.connection.url" value="jdbc:h2:tcp://localhost/~/Download/h2test"/>
          <property name="hibernate.connection.driver_class" value="org.h2.Driver" />
          <property name="hibernate.connection.username" value="sa"/>
          <property name="hibernate.connection.password" value="sa"/>
          <property name="hibernate.show_sql" value="true"/>
          <property name="hibernate.format_sql" value="true"/>
          <property name="hibernate.use_sql_comments" value="true"/>
          <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" />
       </properties>
```

Wildfly datasource configuration

wildflyHome/standalone/configuration/standalone.xml

a datasource refers to a a driver

a driver refers to a module

a module is defined by a module.xml+ jar

```
<subsystem xmlns="urn:jboss:domain:datasources:6.0">
    <datasources>
        <datasource jndi-name="java:jboss/datasources/ExampleDS" pool-name="ExampleDS" enabled="true"
            use-java-context="true"
            statistics-enabled="${wildfly.datasources.statistics-enabled:${wildfly
            .statistics-enabled:false}}">
            <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE</connection-url>
            <driver>h2</driver>
            <security>
                <user-name>sa</user-name>
                <password>sa</password>
            </security>
        </datasource>
        <datasource jndi-name="java:jboss/datasources/DerbyDS" pool-name="DerbyDS" enabled="true"
            use-ccm="false">
              <connection-url>jdbc:derby://localhost:1527/SESSION_DB;create=true</connection-url>
                  <driver>org.apache.derby</driver>
                            <security>
                                <user-name>user1</user-name>
                                <password>pw</password>
                            </security>
                            <validation>
                                <validate-on-match>false</validate-on-match>
                                <background-validation>false</background-validation>
                            </validation>
                            <statement>
                               <share-prepared-statements>false</share-prepared-statements>
                            </statement>
        </datasource>
        <drivers>
            <driver name="h2" module="com.h2database.h2">
                <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
            </driver>
            <driver name="org.apache.derby" module="org.apache.derby">
                <xa-datasource-class>org.apache.derby.jdbc.ClientXADataSource</xa-datasource-class>
            </driver>
        </drivers>
    </datasources>
</subsystem>
```

standalone.xml

you must change the content
of the datasouces section like this

```
<datasources>
        <datasource …
        </datasource>
        <datasource jndi-name="java:jboss/datasources/DerbyDS" pool-name="DerbyDS" enabled="true" use-ccm="false">
          <connection-url>jdbc:derby://localhost:1527/SESSION_DB;create=true</connection-url>
          <driver>org.apache.derby</driver>
          <security>
            <user-name>user1</user-name>
            <password>pw</password>
          </security>
          <validation>
            <validate-on-match>false</validate-on-match>
            <background-validation>false</background-validation>
          </validation>
          <statement>
            <share-prepared-statements>false</share-prepared-statements>
          </statement>
</datasource>
```

JNDI name

DS Symbolic name

Name of your DB

Drivers symbolic name

```
        <drivers>
          <driver name="h2" module="…h2database.h2">
            <xa-datasource-class>org…2.jdbcx.JdbcDataSource</xa-dat…ource-class>
          </driver>
          <driver name="org.apache.derby" module="org.apache.derby">
            <xa-datasource-class>org.apache.derby.jdbc.ClientXADataSource</xa-datasource-class>
          </driver>
        </drivers>
      </datasources>
```

Drivers symbolic name

Module name

- you must change the name of your DB
- you may change the symbolic name and

the last token of the JNDI name

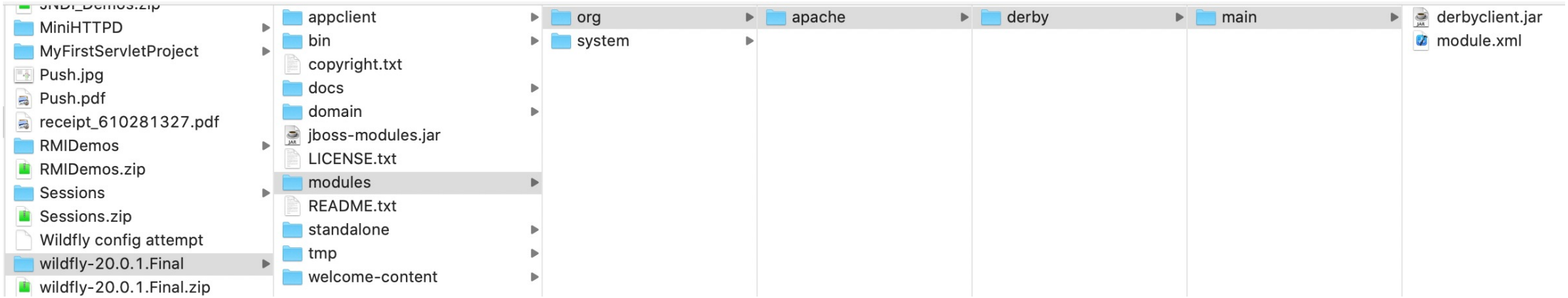
Derby example – not working!

Configure the datasource

- Make sure the driver libraries are included. In your Wildfly home directory, you should have the following directory structure (if not, create it):

modules->org->apache->derby-> main

- In main you should have:
  - derbyclient.jar
  - module.xml



Configure the datasource

Module name

Module.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.3" name="org.apache.derby">
   <resources>
      <resource-root path="derbyclient.jar"/>
   </resources>
   <dependencies>
      <module name="javax.api"/>
      <module name="javax.transaction.api"/>
   </dependencies>
</module>
```

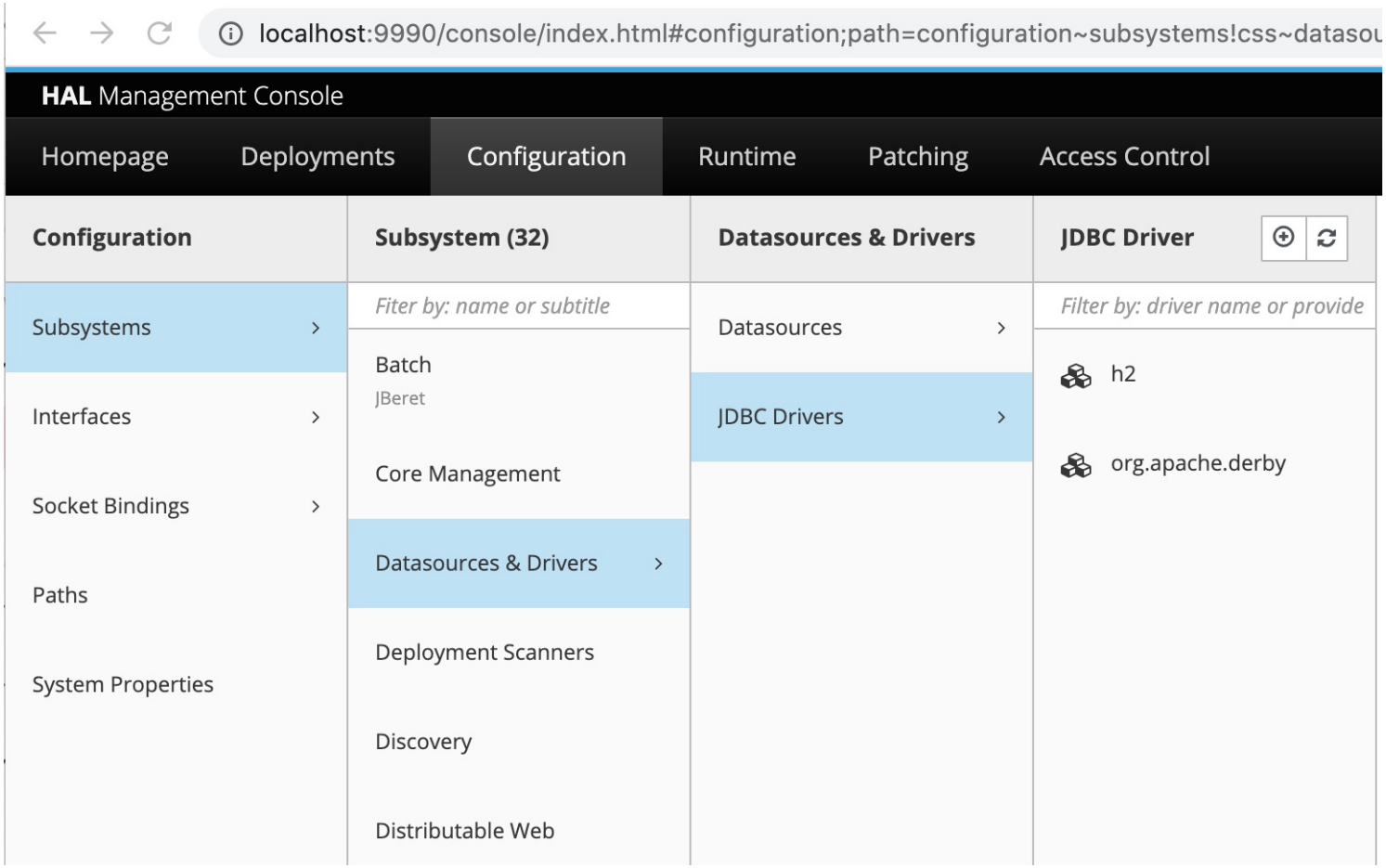
See https://www.hameister.org/JBoss_DatasourceDerby.html
but Derby seems to have a problem right now!

Derby example – not working! *(because the Derby driver is split into multiple jars)*

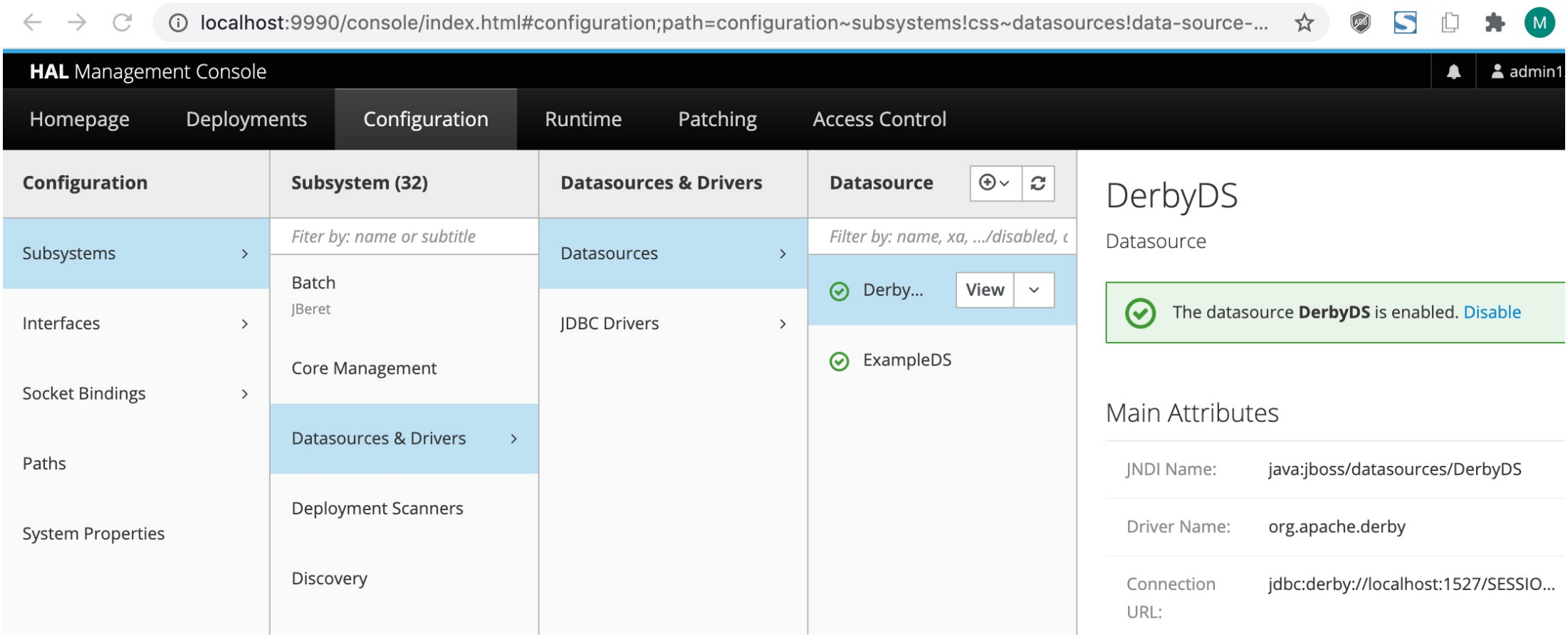
Check the datasource

- Make sure the DB Service is started (e.f. from within Netbeans)
- Open a shell, and cd to the bin directory of wildfly.
- Start wildfly
- Check it in the console

Connect to the console, check the driver



Connect to the console, check the datasource



Introduction to Entities

Entities

- Entities have a client-visible, persistent *identity* (the primary key) that is distinct from their object reference.
- Entities have persistent, client-visible *state*.
- Entities are *not remotely accessible*.
- An entity's *lifetime* may be completely independent of an application's lifetime.
- Entities can be used in both Java EE and J2SE environments

Entities - example

This demo entity represents a Bank Account.

The entity is NOT a remote object and can ONLY be accessed locally by clients.

However, it is made serializable so that instances can be passed by value to remote clients as DETACHED ENTITY

```
package examples.entity.intro;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
@Entity
public class Account implements Serializable {
    // The account number is the primary key
    @Id
    public int accountNumber;
    public int balance;
    private String ownerName;
    String getOwnerName() {return ownerName;}
    void setOwnerName(String s) {ownerName=s;}

    /** Entities must have a public no-arg constructor */
    public Account() {
        // our own simple primary key generation
        accountNumber = (int) System.nanoTime();
    }
```

Entities - example

```
    public void deposit(int amount) {
      balance += amount;
    }
    public int withdraw(int amount) {
        if (amount > balance) {
          return 0;
        } else {
          balance -= amount;
          return amount;
        }
    }
}
```

The entity can expose **business methods**, such as a method to decrease a bank account balance, to manipulate or access that data.

Like a session bean class, an entity class can also declare some standard callback methods or a callback listener class.

Access to the entity's persistent state is by direct field access. An entity's state can also be accessed using JavaBean-style set and get methods.

The persistence provider can determine which access style is used by looking at how annotations are applied. In the discussed example the @Id annotation is applied to a field (as opposed to annotation applied to a method, so we have field access).

Access to the Entity

```
package examples.entity.intro;
import java.util.List;
import javax.ejb.Stateless;
import javax.ejb.Remote;
import javax.persistence.PersistenceContext;
import javax.persistence.EntityManager;
import javax.persistence.Query;
```

```
@Stateless
@Remote(Bank.class)
public class BankBean implements Bank {
  @PersistenceContext
  private EntityManager manager;
  public List<Account> listAccounts() {
      Query query = manager.createQuery ("SELECT a FROM Account a");  return
      query.getResultList();
  }
    public Account openAccount(String ownerName) {  Account account =
    new Account();  account.ownerName = ownerName;
    manager.persist(account);
      return account;
  }
```

Access to the Entity

```
    public int getBalance(int accountNumber) {
        Account account = manager.find(Account.class, accountNumber);
        return account.balance;
    }
    public void deposit(int accountNumber, int amount) {
        Account account = manager.find(Account.class, accountNumber);
        account.deposit(amount);
    }
    public int withdraw(int accountNumber, int amount) {
        Account account = manager.find(Account.class, accountNumber);
        return account.withdraw(amount);
    }
    public void close(int accountNumber) {
        Account account = manager.find(Account.class, accountNumber);
        manager.remove(account);
    }
}
```

Persistence.xml

```
<?xml version= 1.0 encoding= UTF-8 ?>
<persistence xmlns= http://java.sun.com/xml/ns/persistence >
    <persistence-unit name= intro />
</persistence>
```

- A persistence unit is defined in a special descriptor file, the persistence.xml file, which is simply added to the META-INF directory of an arbitrary archive, such as an Ejb-jar, .ear, or .war file, or in a plain .jar file.

datasource configuration on Wildfly

http://www.mastertheboss.com/jboss-server/jboss-datasource/how-to-configure-a-datasource-with-jboss-7

Simple working example
with stateless + entity
and web client

Simple working example
with stateless + entity
and web client

Configure the server

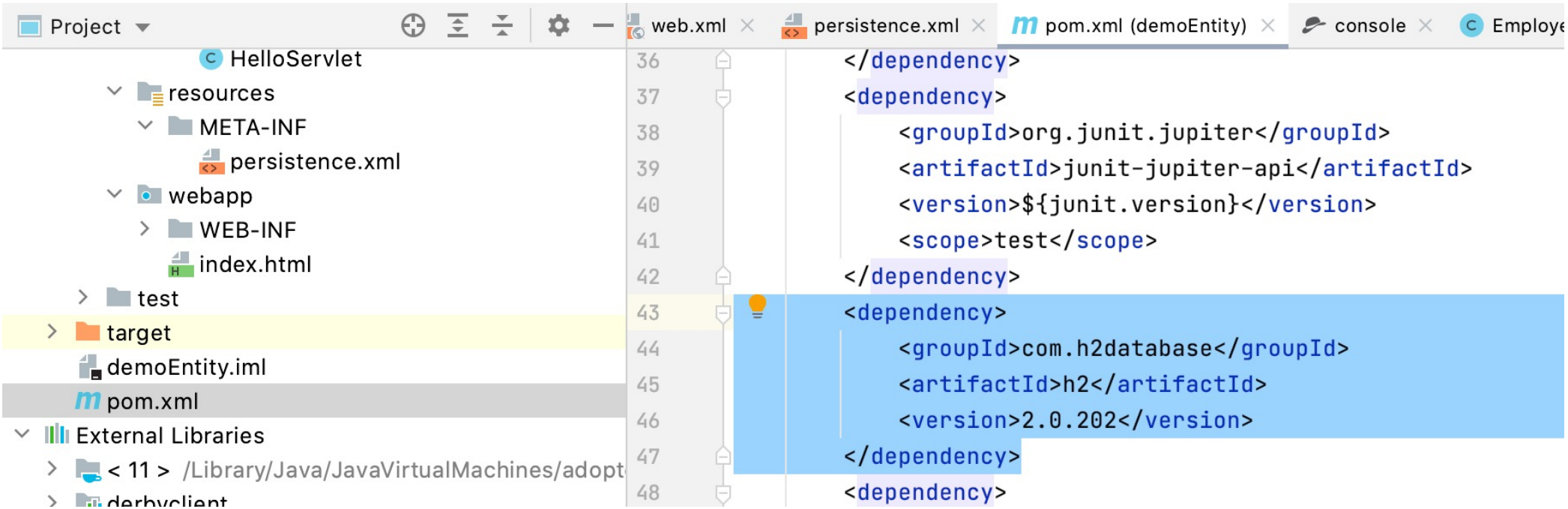
In the standalone.xml, add the following section (in the datasource section)

```
<datasource jndi-name="java:jboss/datasources/TestDS" pool-name="TestDS" enabled="true" use-java-context="true"
    statistics-enabled="${wildfly.datasources.statistics-enabled:${wildfly.statistics-enabled:false}}">
    <connection-url>jdbc:h2:tcp://localhost/~/Download/h2test;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE</connection-url>
    <driver>h2</driver>
    <security>
        <user-name>sa</user-name>
        <password>sa</password>
    </security>
</datasource>
```

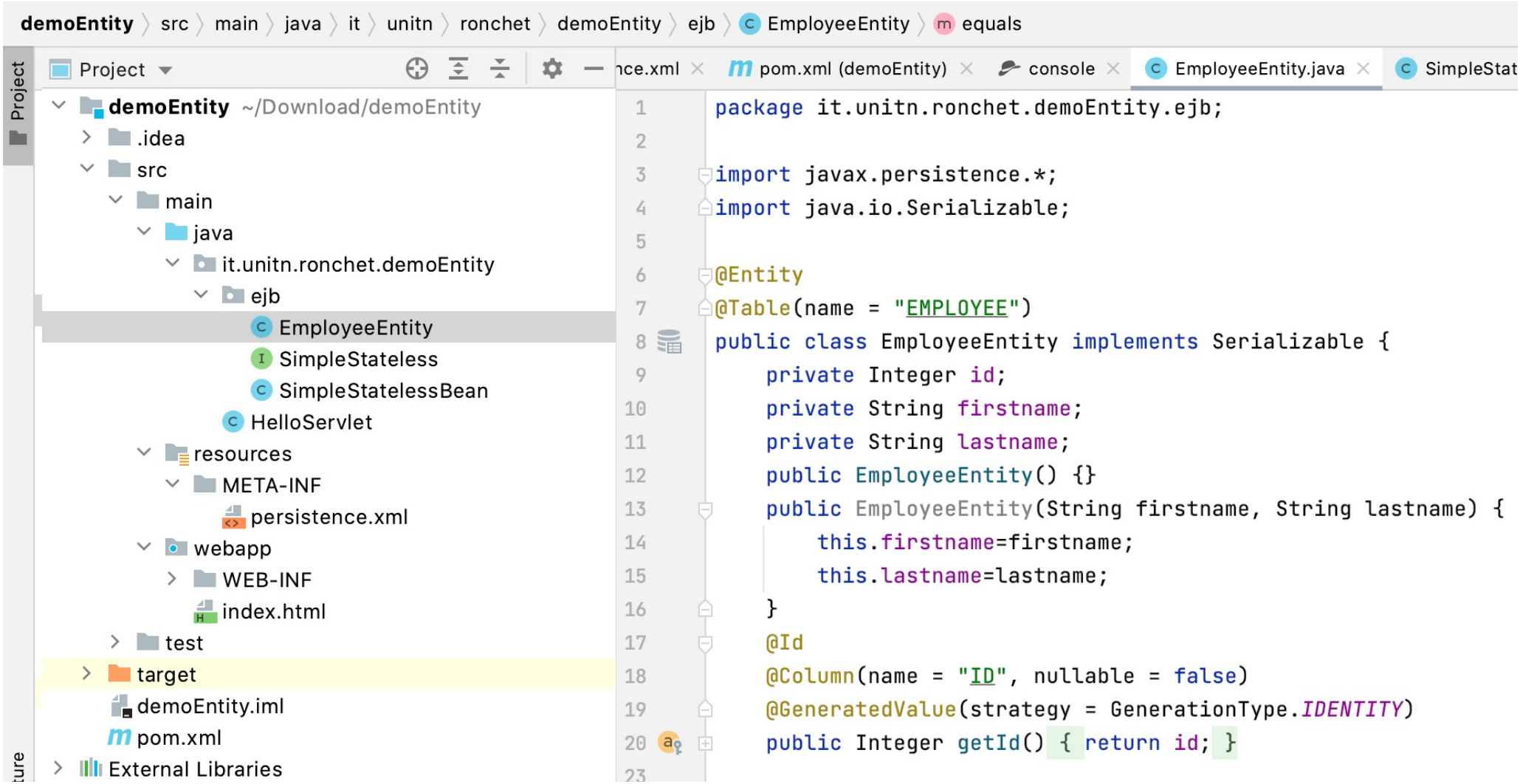
make sure the URL corresponds to the right location in your machine!

pom.xml

Create a JEE project in IntelliJ, choose Wildfly as server, and configure the dependency on the DB library:



The entity (in the EJB server)



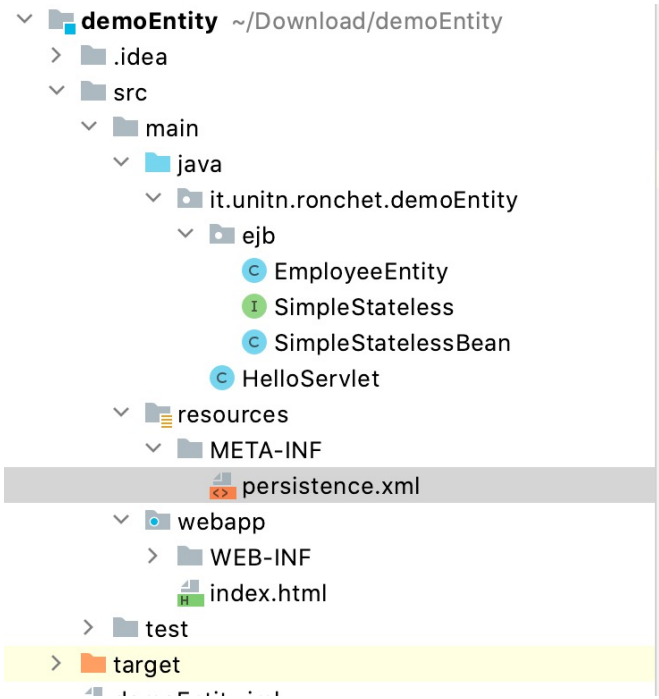
The entity (in the EJB server) – part 2

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

@Basic
@Column(name = "FIRSTNAME", nullable = true, length = 30)
public String getFirstname() { return firstname; }

public void setFirstname(String firstname) { this.firstname = firstname; }

@Basic
@Column(name = "LASTNAME", nullable = true, length = 30)
public String getLastname() { return lastname; }

public void setLastname(String lastname) { this.lastname = lastname; }
```

The entity (in the EJB server) – part 3

```
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        EmployeeEntity that = (EmployeeEntity) o;

        if (id != null ? !id.equals(that.id) : that.id != null) return false;
        if (firstname != null ? !firstname.equals(that.firstname) : that.firstname != null) return false;
        if (lastname != null ? !lastname.equals(that.lastname) : that.lastname != null) return false;

        return true;
    }

    @Override
    public int hashCode() {
        int result = id != null ? id.hashCode() : 0;
        result = 31 * result + (firstname != null ? firstname.hashCode() : 0);
        result = 31 * result + (lastname != null ? lastname.hashCode() : 0);
        return result;
    }
}
```

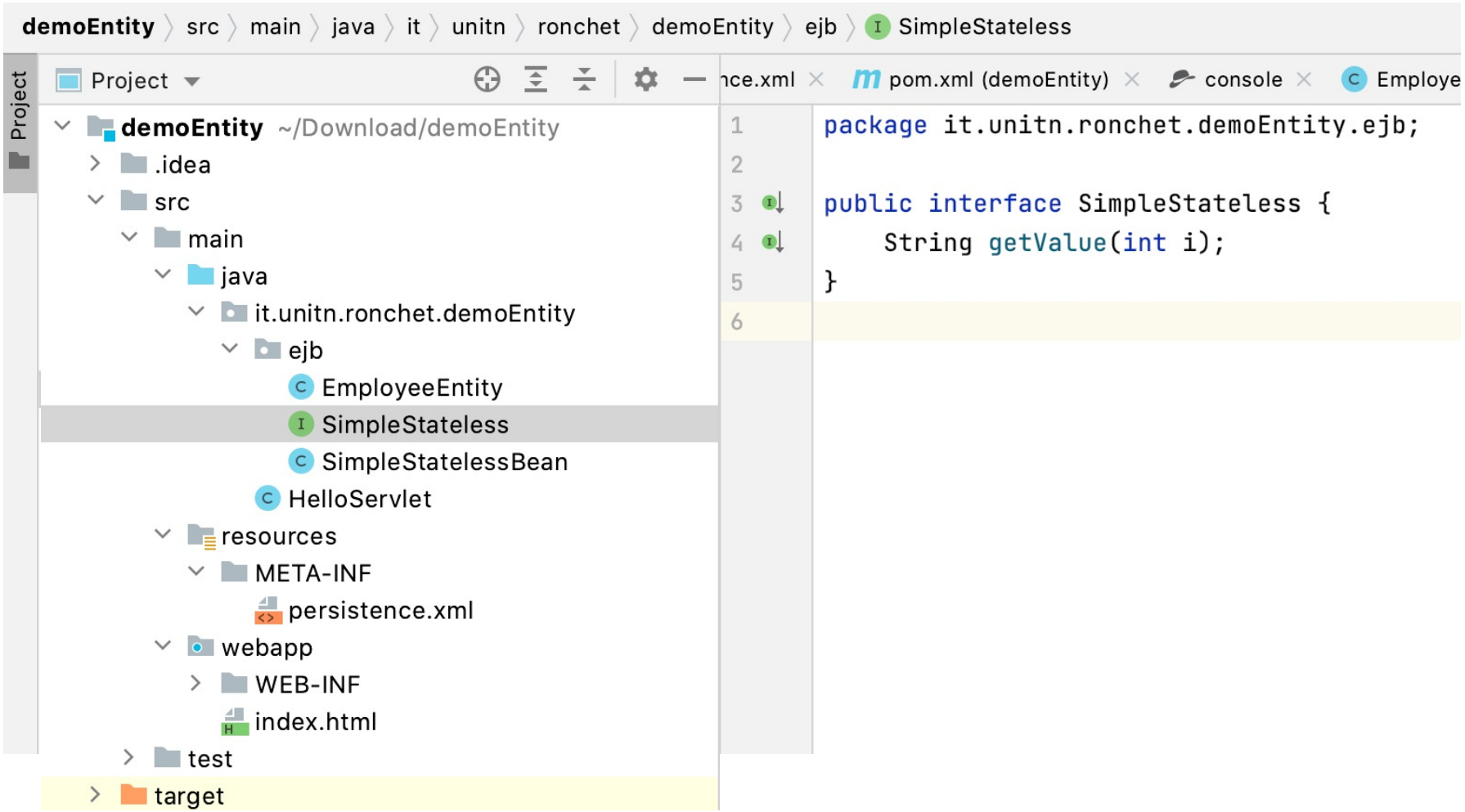
Persistence.xml



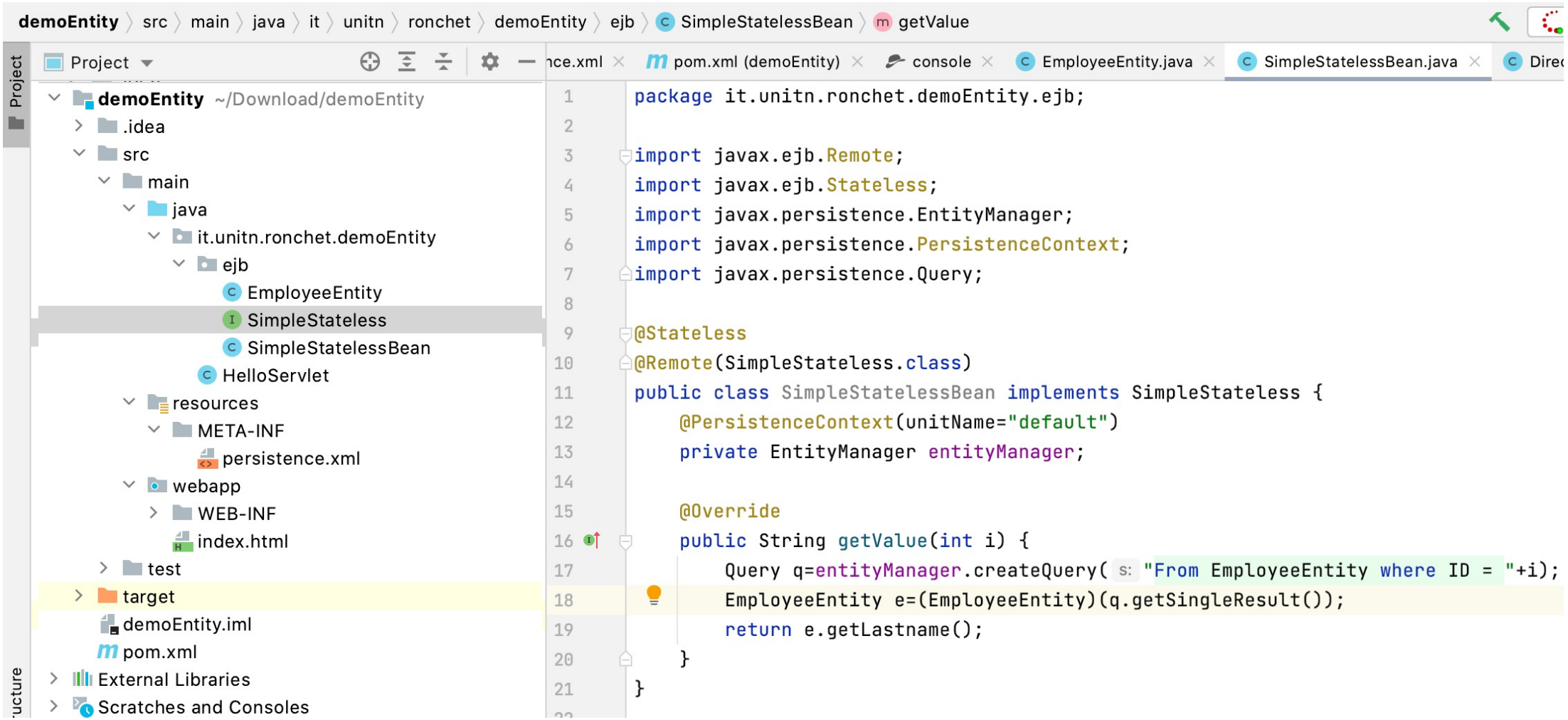

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                                 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
             version="2.2">
    <persistence-unit name="default">
        <jta-data-source>java:jboss/datasources/TestDS</jta-data-source>
    </persistence-unit>
</persistence>
```

The remote interface

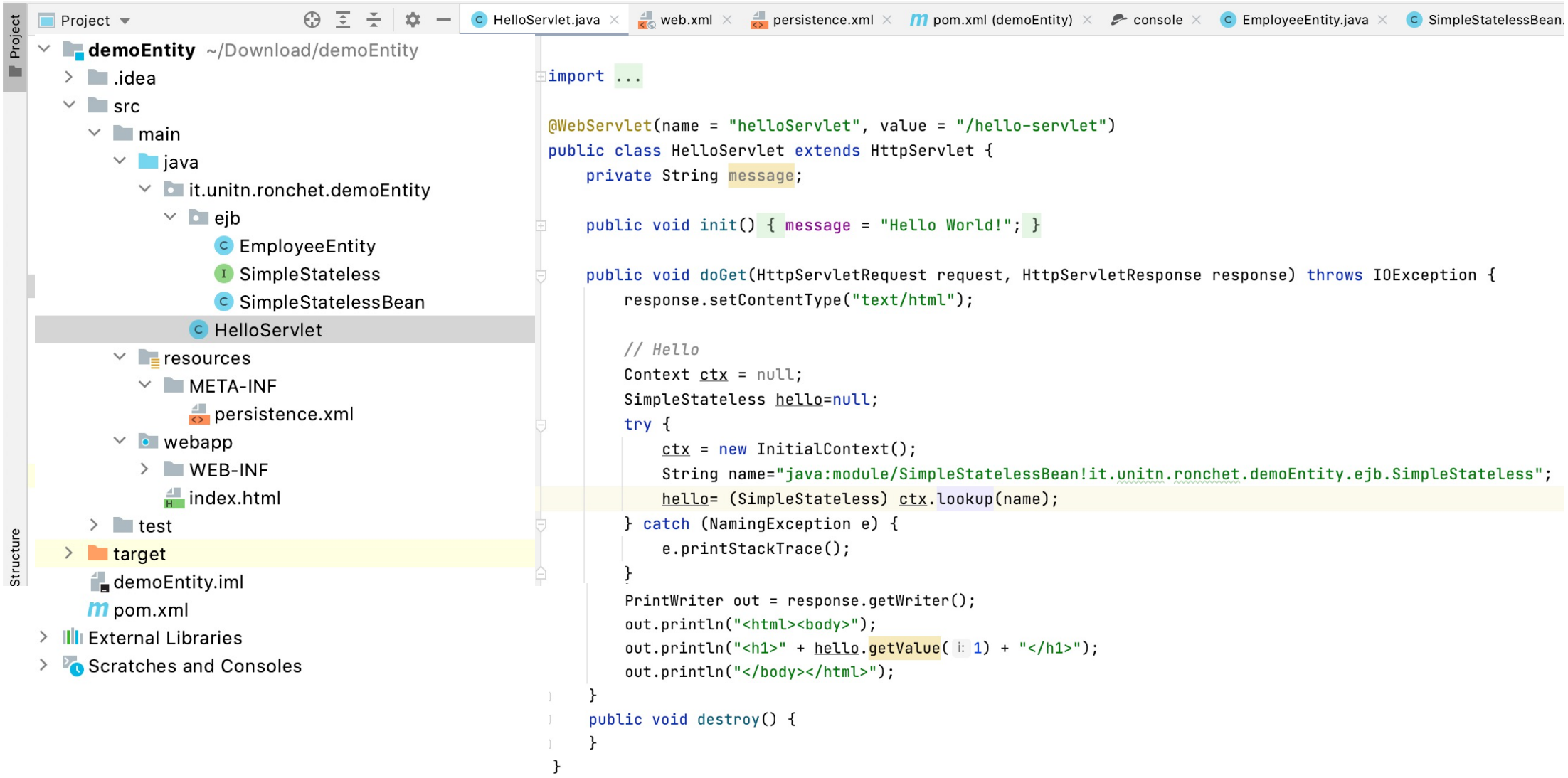
included both il the client and in the server!



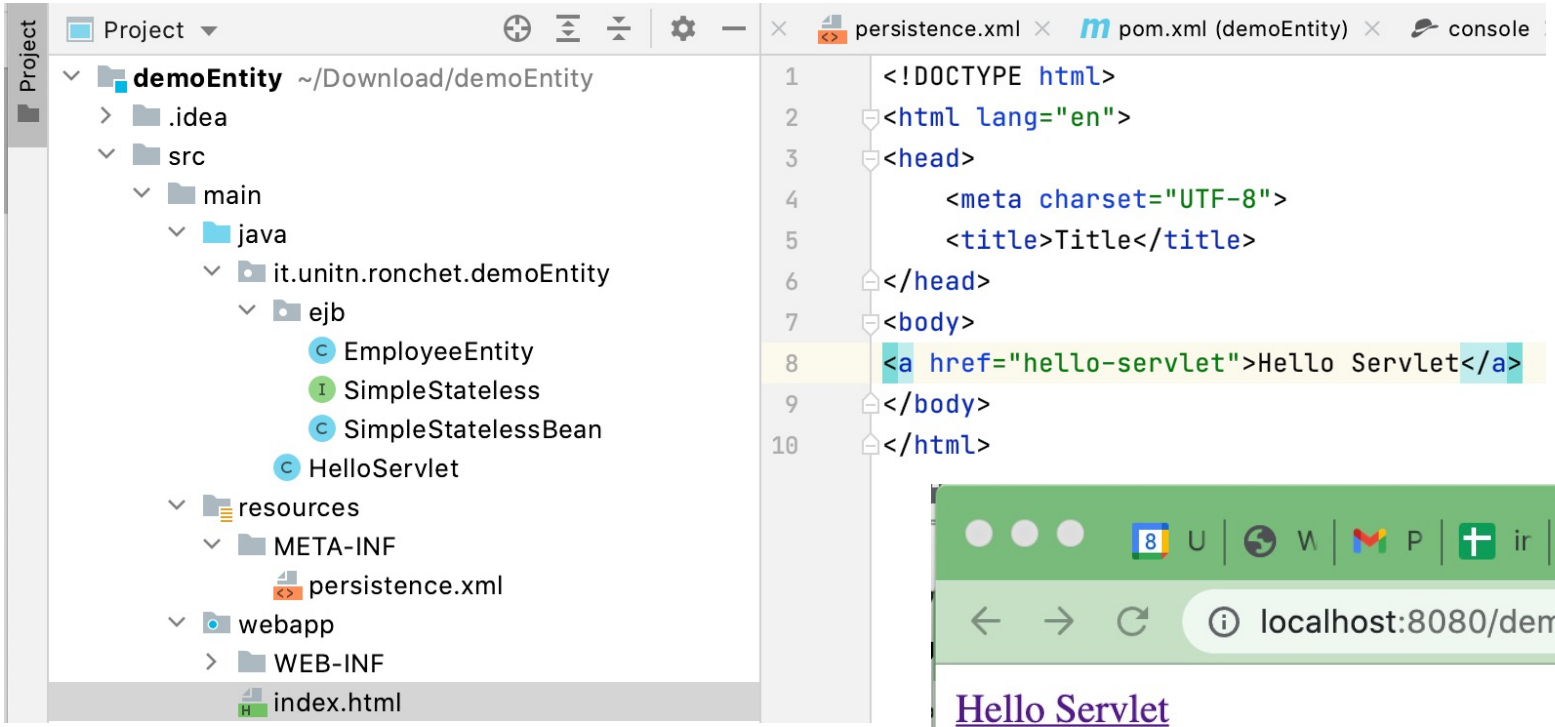
The stateless bean (in the EJB server)

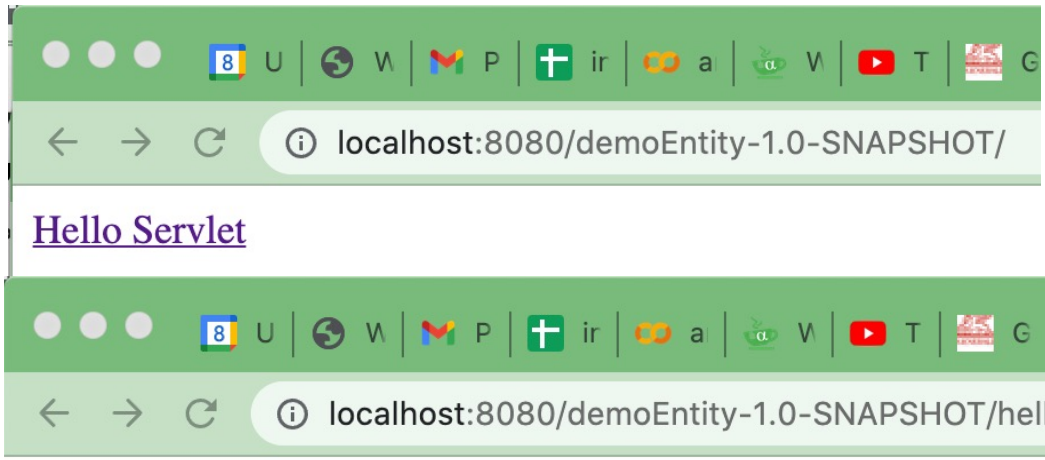


The servlet (in the Web server)



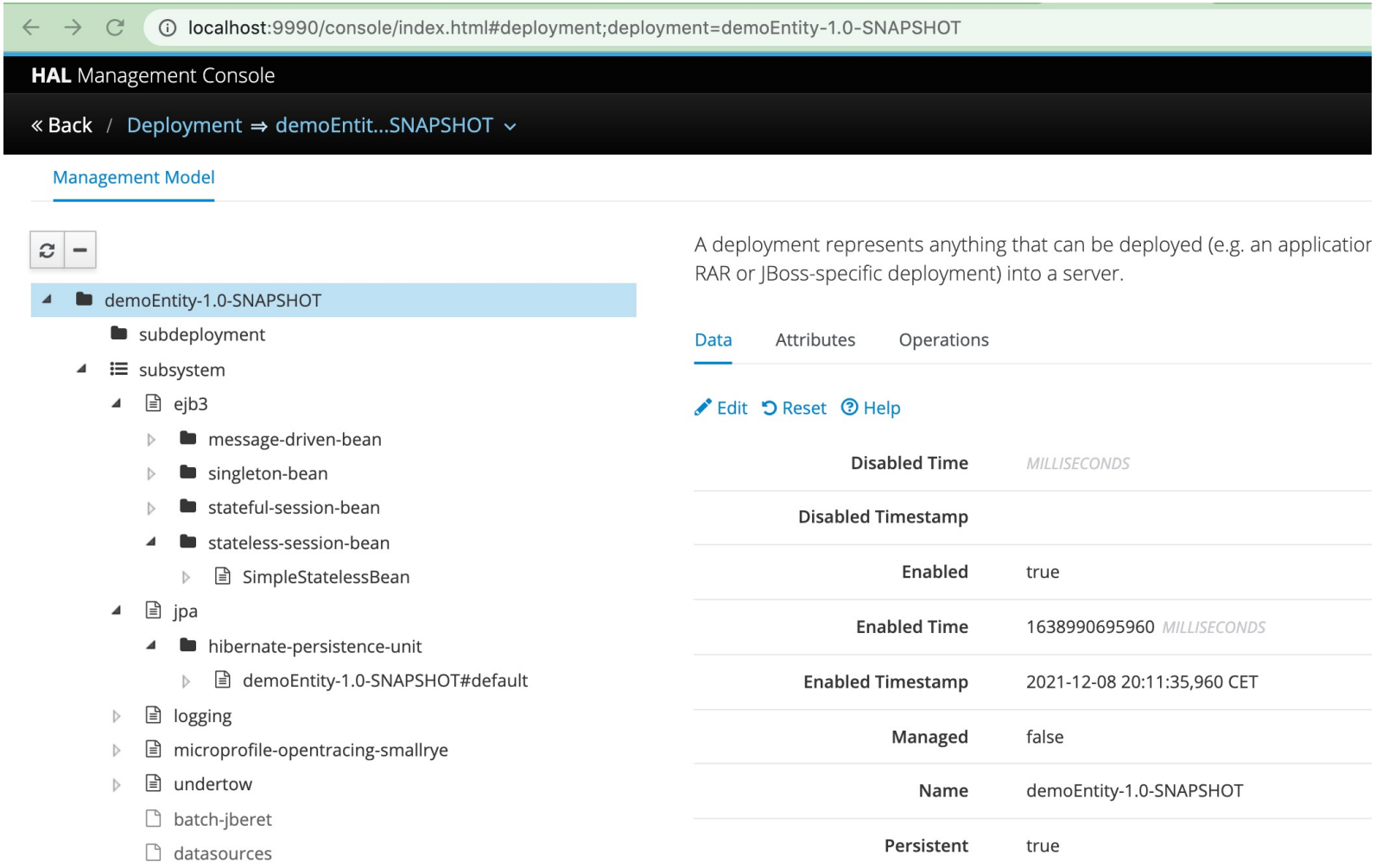
index.html (in the Web server)





Rossi

On the console, let's check the deployment



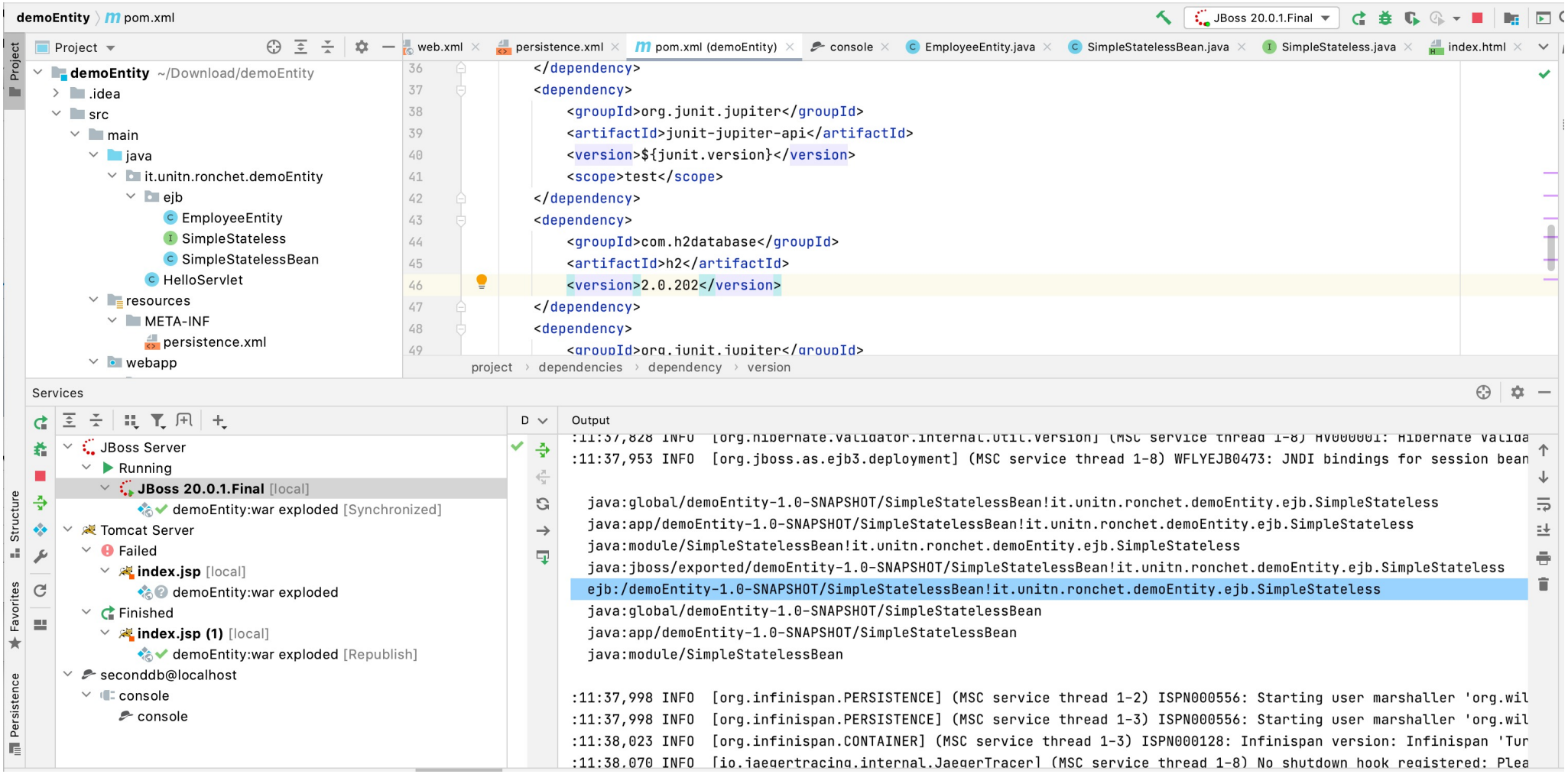
Hint: How do I find the right JNDI name?

Alternative 1: look at the starting log of the server, choose the "ejb" one

```
:11:37,828 INFO  [org.hibernate.validator.internal.util.version] (MSC service thread 1-8) HV000001: Hibernate Valida
:11:37,953 INFO  [org.jboss.as.ejb3.deployment] (MSC service thread 1-8) WFLYEJB0473: JNDI bindings for session bear

  java:global/demoEntity-1.0-SNAPSHOT/SimpleStatelessBean!it.unitn.ronchet.demoEntity.ejb.SimpleStateless
  java:app/demoEntity-1.0-SNAPSHOT/SimpleStatelessBean!it.unitn.ronchet.demoEntity.ejb.SimpleStateless
  java:module/SimpleStatelessBean!it.unitn.ronchet.demoEntity.ejb.SimpleStateless
  java:jboss/exported/demoEntity-1.0-SNAPSHOT/SimpleStatelessBean!it.unitn.ronchet.demoEntity.ejb.SimpleStateless
  ejb:/demoEntity-1.0-SNAPSHOT/SimpleStatelessBean!it.unitn.ronchet.demoEntity.ejb.SimpleStateless
  java:global/demoEntity-1.0-SNAPSHOT/SimpleStatelessBean
  java:app/demoEntity-1.0-SNAPSHOT/SimpleStatelessBean
  java:module/SimpleStatelessBean

:11:37,998 INFO  [org.infinispan.PERSISTENCE] (MSC service thread 1-2) ISPN000556: Starting user marshaller 'org.wil
:11:37,998 INFO  [org.infinispan.PERSISTENCE] (MSC service thread 1-3) ISPN000556: Starting user marshaller 'org.wil
:11:38,023 INFO  [org.infinispan.CONTAINER] (MSC service thread 1-3) ISPN000128: Infinispan version: Infinispan 'Tur
```

The one we need is: ejb:/demoEntity-1.0-SNAPSHOT/SimpleStatelessBean!it.unitn.ronchet.demoEntity.ejb.SimpleStateless

Check the log, and note the JNDI bindings



How do I find the right JNDI name?

https://docs.jboss.org/author/display/WFLY10/EJB%20invocations%20from%20a%20remote%20client%20using%20JNDI.html

Alternative 2: Compose it following a rule:

```
// The app name is the application name of the deployed EJBs. This is typically the ear name
// without the .ear suffix. However, the application name could be overridden in the application.xml
// of the EJB deployment on the server.
// If haven't deployed the application as a .ear, the app name for us will be an empty string
        final String appName = "";
// This is the module name of the deployed EJBs on the server. This is typically the jar name of the
// EJB deployment, without the .jar suffix, but can be overridden via the ejb-jar.xml
// In this example, we have deployed the EJBs in a jboss-as-ejb-remote-app.jar, so the module name is
// jboss-as-ejb-remote-app
        final String moduleName = "jboss-as-ejb-remote-app";
// AS7 allows each deployment to have an (optional) distinct name. We haven't specified a distinct name for
// our EJB deployment, so this is an empty string
        final String distinctName = "";
// The EJB name which by default is the simple class name of the bean implementation class
        final String beanName = CalculatorBean.class.getSimpleName();
// the remote view fully qualified class name -
// add a ?stateful string as the last part of the jndi name for stateful bean lookup
        final String viewClassName = RemoteCalculator.class.getName();
// let's do the lookup
        return (RemoteCalculator) context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName);
```

Simple working example
with stateless + entity
and desktop client

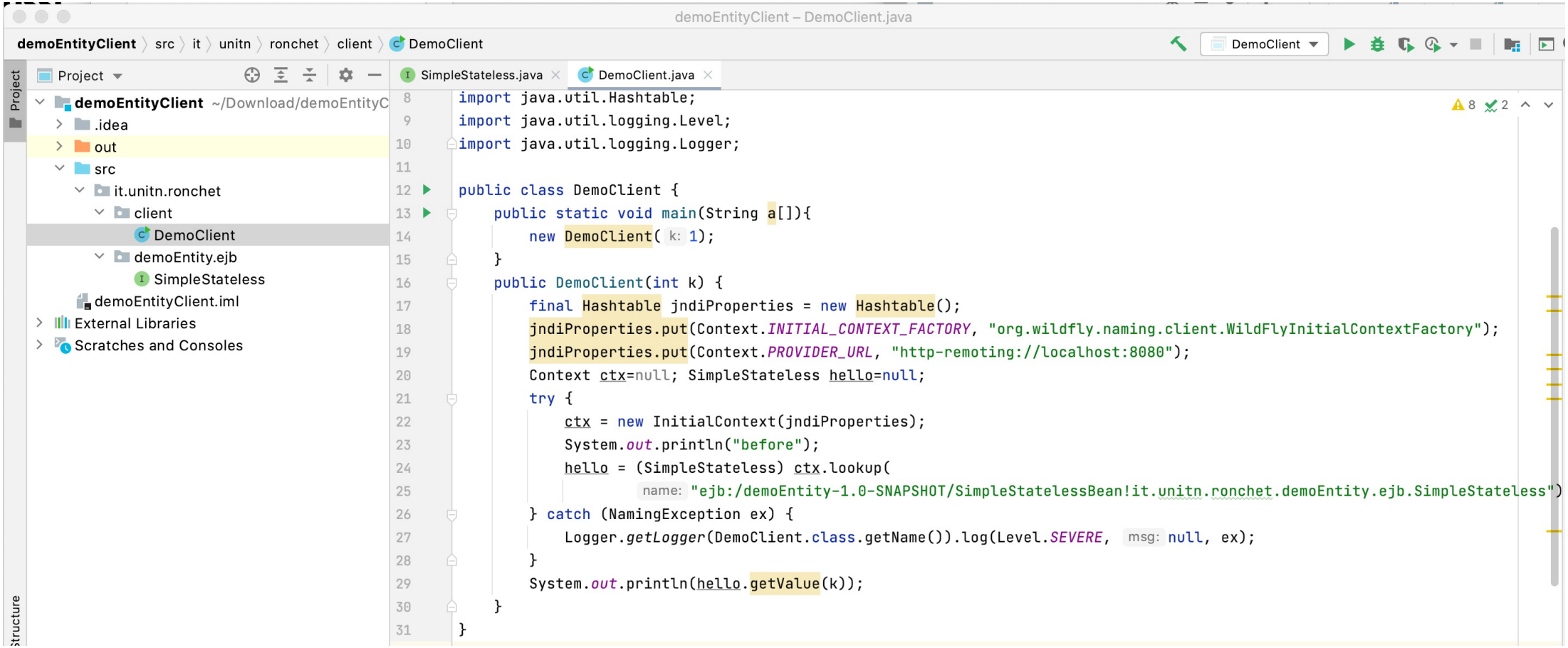
Let's create the client project

We need to create a standard Java project (not JEE):

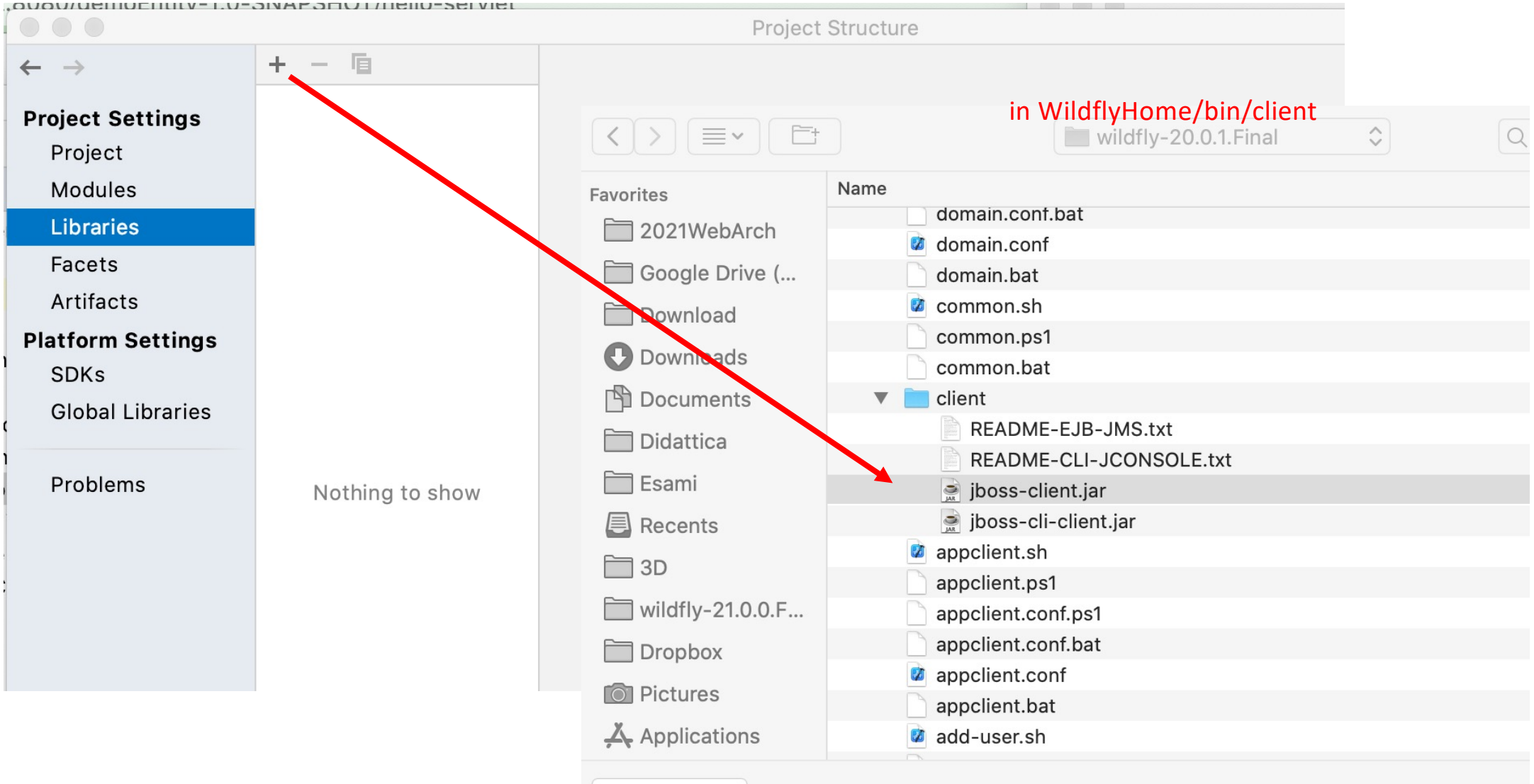
- copy in it the Remote interface
- write the client code (using the right JNDI bindings):

```
final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY,
       "org.wildfly.naming.client.WildFlyInitialContextFactory"
);

jndiProperties.put(Context.PROVIDER_URL,
       "http-remoting://localhost:8080");

try {ctx = new InitialContext(jndiProperties);}

catch (NamingException ex) {ex.printStackTrace();}
```

Let's create the client project

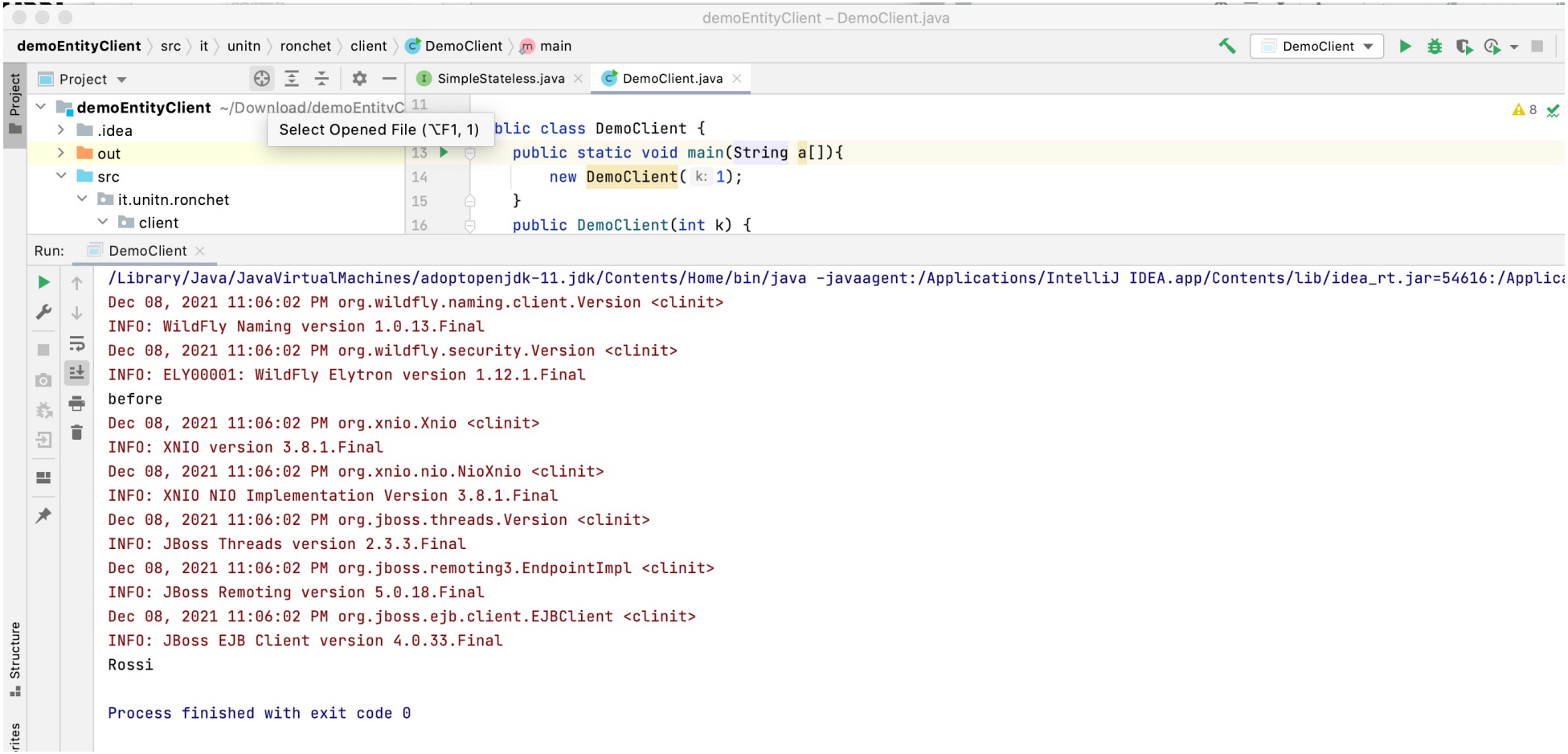


Add libraries to the client project



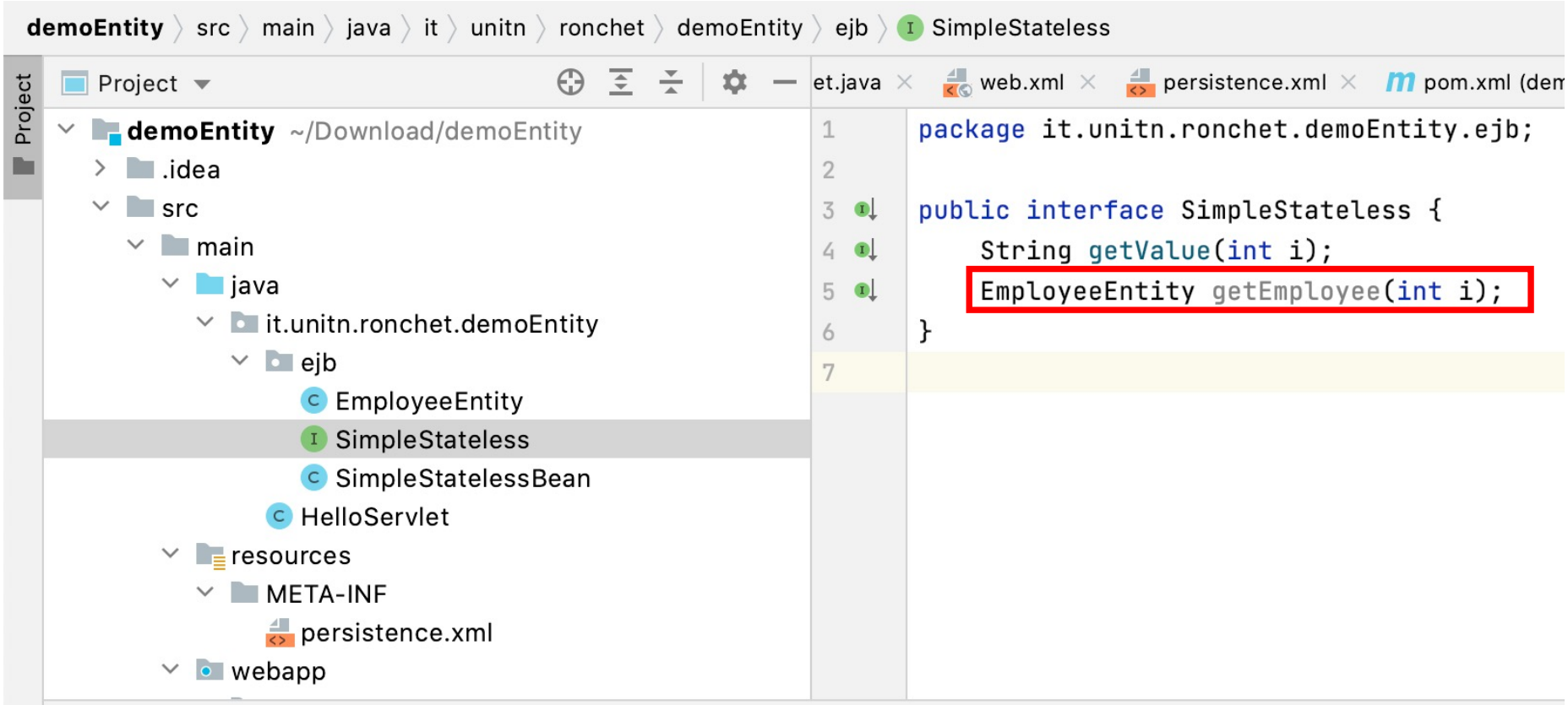
Now we can run the client project

Of course, the server part must be deployed first.

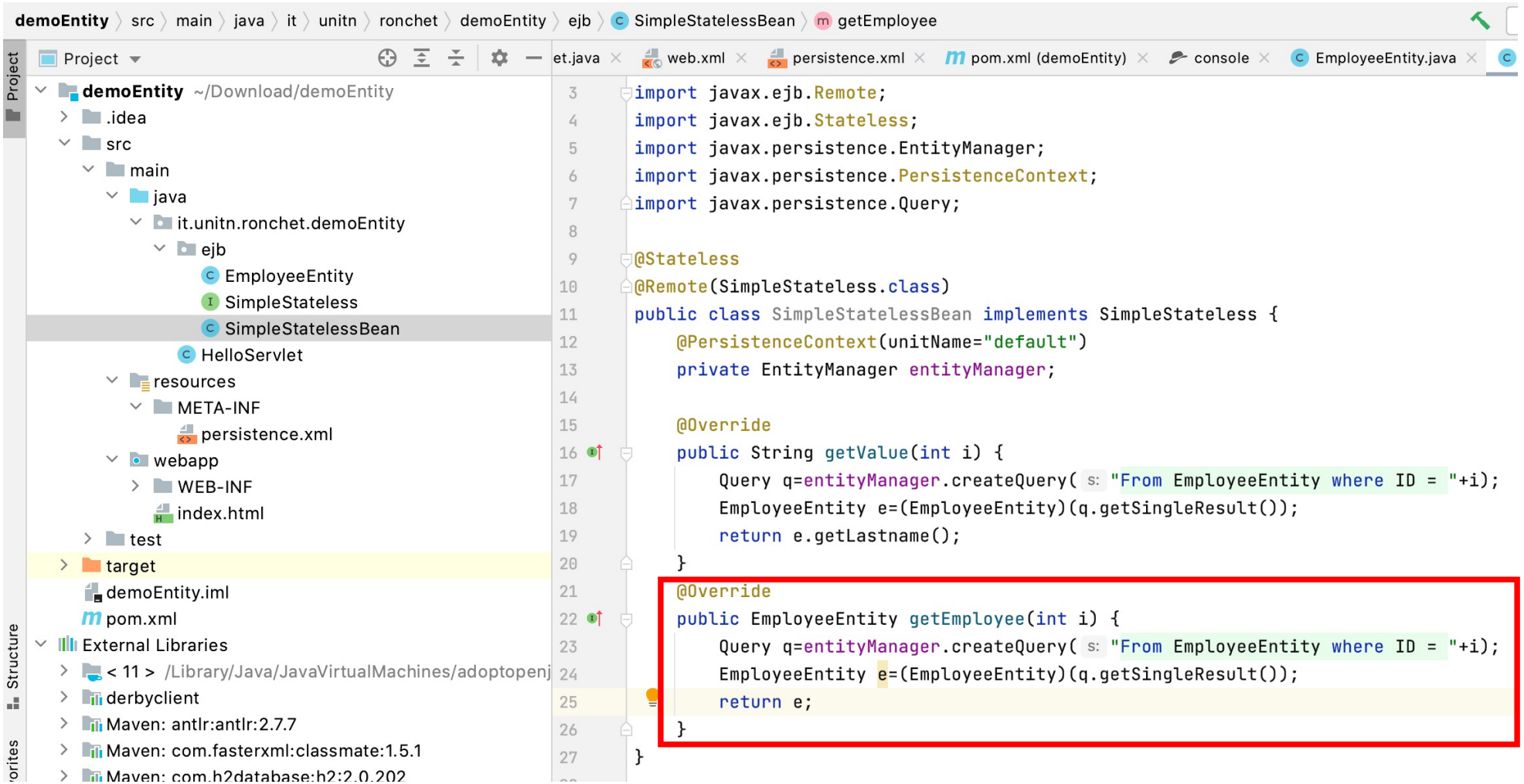


Simple working example with Detached Entity as DTO

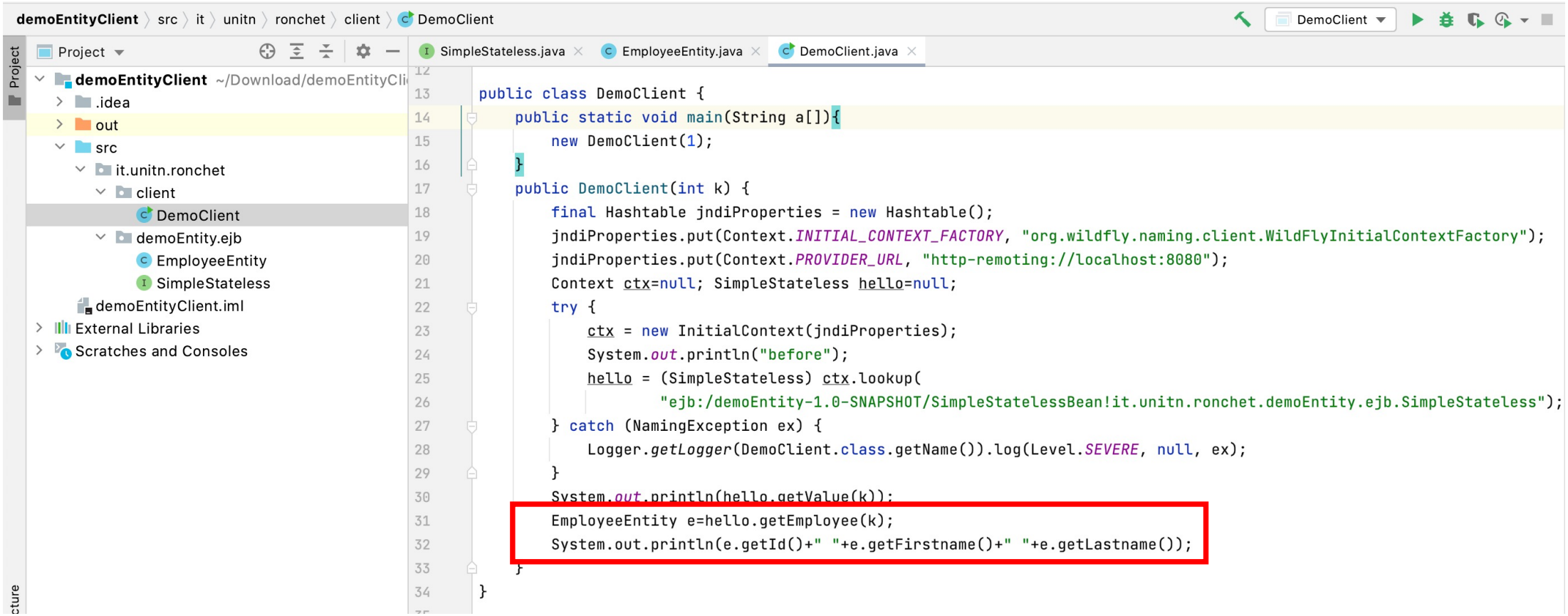
Modified server project



Modified server project

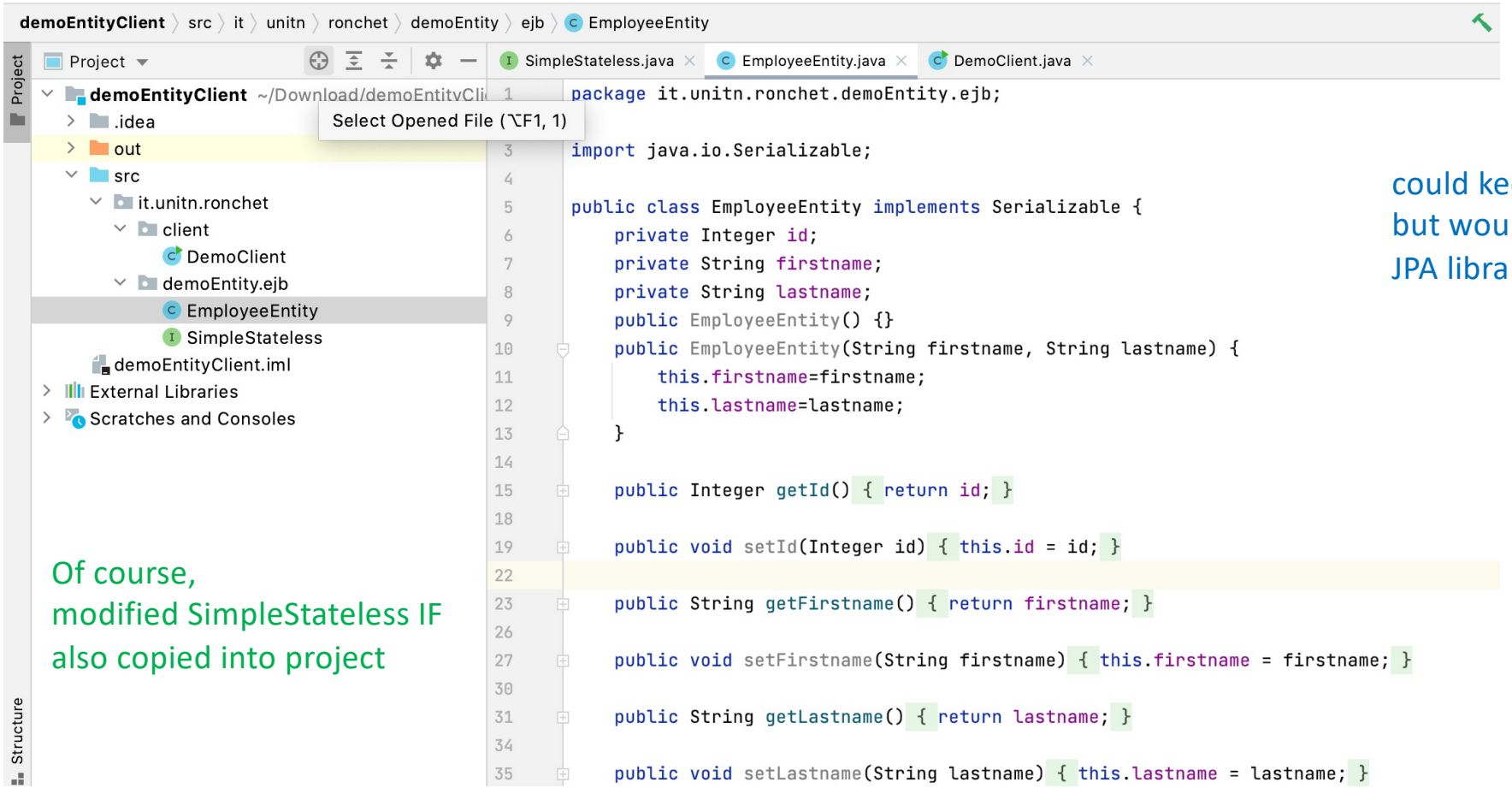


Modified client project



Modified client project

copied without annotations

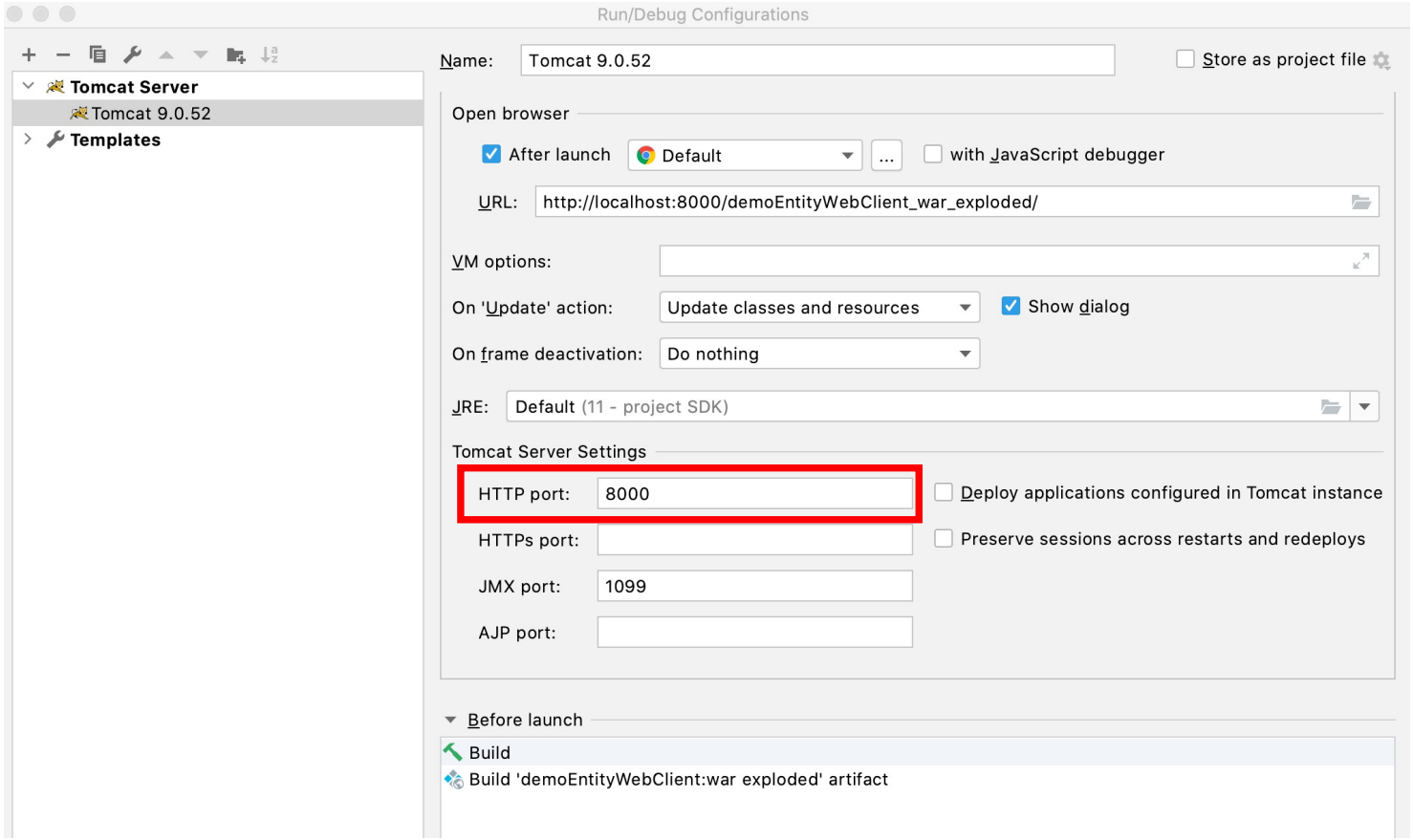


could keep annotations,
but would need to include
JPA libraries

Of course,
modified SimpleStateless IF
also copied into project

Simple working example with External Web Server

Set up project

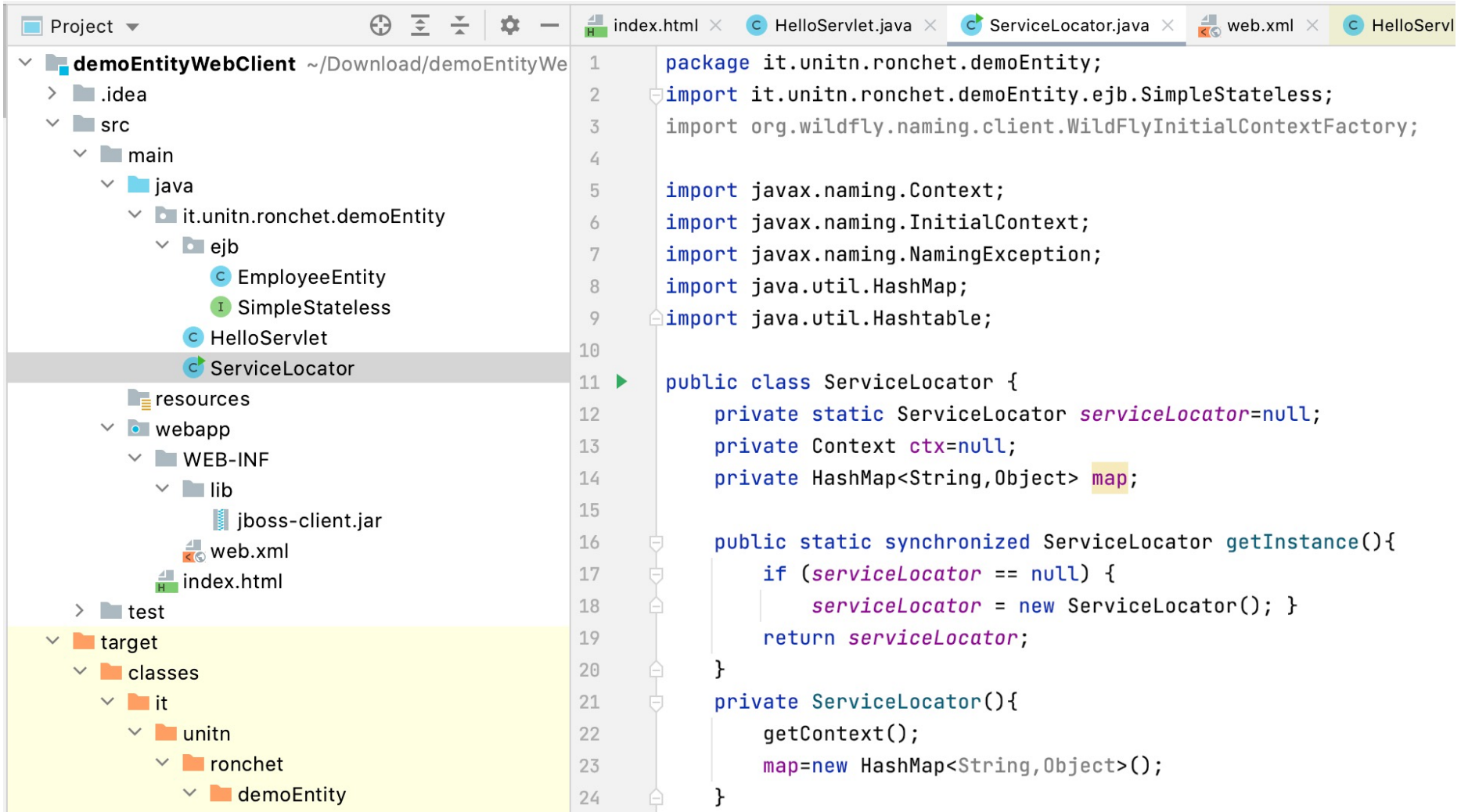


- create new Web project
- choose Tomcat deployment
- configure Tomcat on a port not used by Wildfly
- add JBoss client library to project

Servlet


demoEntityWebClient – HelloServlet.java
demoEntityWebClient 〉 src 〉 main 〉 java 〉 it 〉 unitn 〉 ronchet 〉 demoEntity 〉 HelloServlet 〉 doGet
Tomcat 9.0.52
Project
index.html
HelloServlet.java
ServiceLocator.java
web.xml
HelloServlet.class
EmployeeEntity.java
SimpleStateless.java
demoEntityWebClient ~/Download/demoEntityWe
.idea
src
main
java
it.unitn.ronchet.demoEntity
ejb
EmployeeEntity
SimpleStateless
HelloServlet
ServiceLocator
resources
webapp
WEB-INF
lib
jboss-client.jar
web.xml
index.html
test
12 import java.io.PrintWriter;
13 import java.util.Hashtable;
14
15 @WebServlet(name = "helloServlet", value = "/hello-servlet")
16 public class HelloServlet extends HttpServlet {
17
18 public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
19 response.setContentType("text/html");
20 String handleKey="ejb:/demoEntity-1.0-SNAPSHOT/SimpleStatelessBean!it.unitn.ronchet.demoEntity.ejb.SimpleStateless"
21 SimpleStateless ss=(SimpleStateless) ServiceLocator.getInstance().getHandle(handleKey);
22 PrintWriter out = response.getWriter();
23 out.println("<html><body>");
24 out.println("<h1>" +ss.getValue(i: 1)+ "</h1>");
25 EmployeeEntity e=ss.getEmployee(i: 2);
26 out.println("<h1>" + e.getLastname()+" "+e.getFirstname() + "</h1>");
27 out.println("</body></html>");
28 }
29 }

Service Locator – part 1



Service Locator – part 2