



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

#shutdown server

java -jar \$DERBY_HOME/lib/derbyrun.jar server shutdown

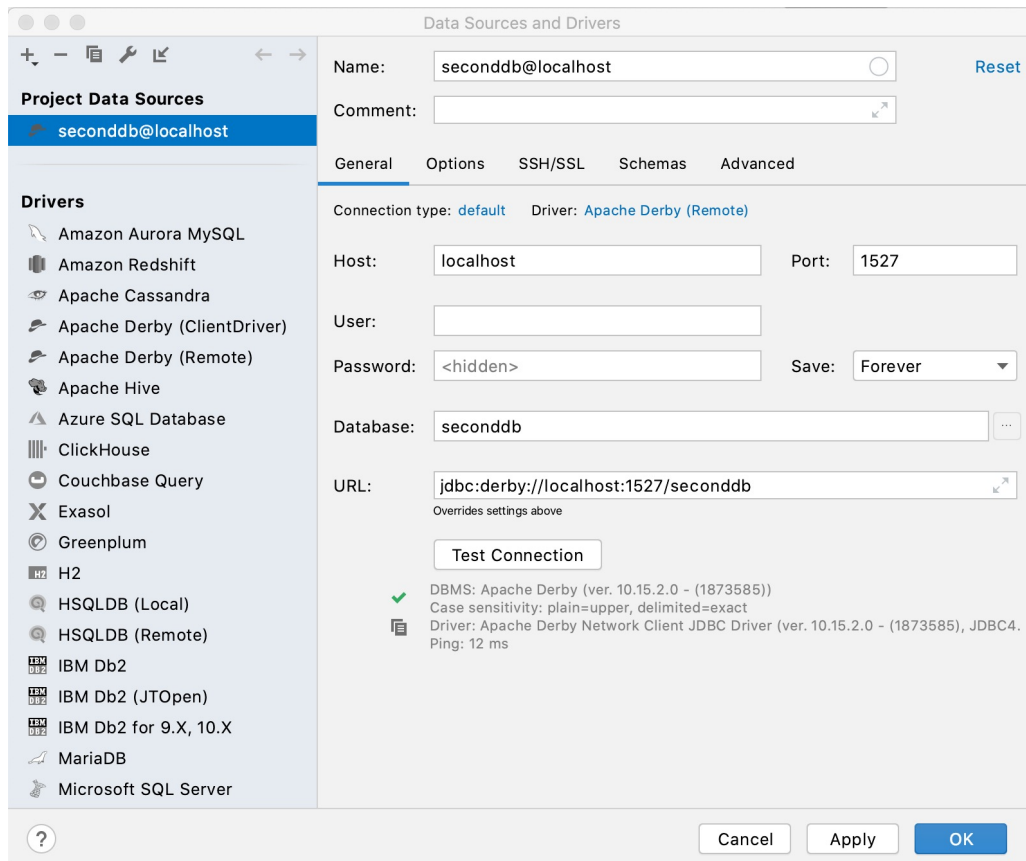
Interacting with the (EMBEDDED) server using ij

```
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;
```

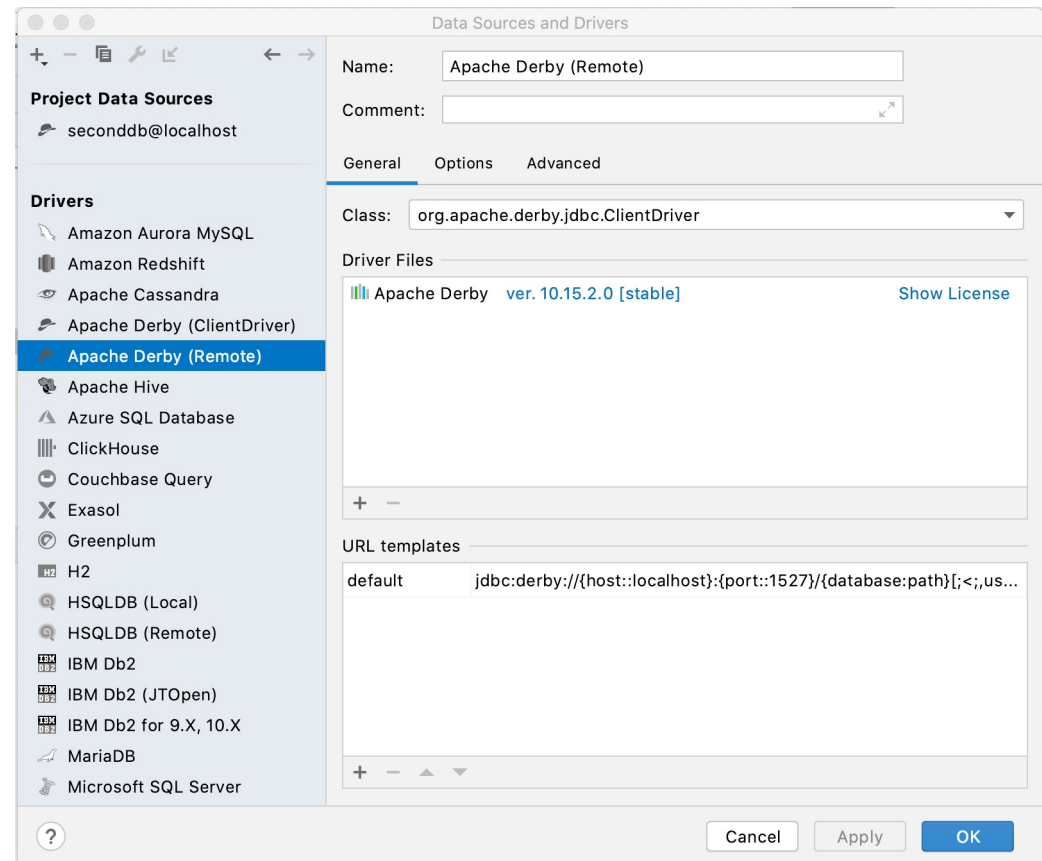

Interacting with the (NETWORK) server using ij

```
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;
```

Viewing it from IntelliJ



In a clone of project demoJPA:



Viewing it from IntelliJ

The screenshot shows the IntelliJ IDEA interface with the 'demoJPAExternalDB - EMPLOYEE' window. The 'Database' tool window is open, displaying the 'seconddb@localhost' connection. The 'EMPLOYEE' table is selected, and its data is shown in a table view. The table has columns: ID (primary key), FIRSTNAME, and LASTNAME. The data rows are:

	ID	FIRSTNAME	LASTNAME
1	1	Valentino	Rossi
2	2	Sofia	Goggia

The 'Database' tool window also shows the schema for the 'EMPLOYEE' table:

- ID: INTEGER (auto increment) = AUTOINCREMENT: sta.
- FIRSTNAME: VARCHAR(30)
- LASTNAME: VARCHAR(30)
- EMPLOYEE_PK (ID): Primary Key
- SQL0000000001-341cc09e-017d-8f42-f51c-fff

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"/>
</properties>
```

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



[Translate](#)

Search:

[Home](#)

[Download](#)

[Cheat Sheet](#)

[Documentation](#)

Downloads

Version 2.0.202 (2021-11-25)

[Windows Installer](#) (SHA1 checksum: f6f6f91c67075a41ce05bdfc4499ee987dacb02e)

[Platform-Independent Zip](#) (SHA1 checksum: e4a6c2e54332304cb4acbe48b55f9421c7f4b870)

Version 1.4.200 (2019-10-14), Last Stable

[Windows Installer](#)

[Platform-Independent Zip](#)

Archive Downloads

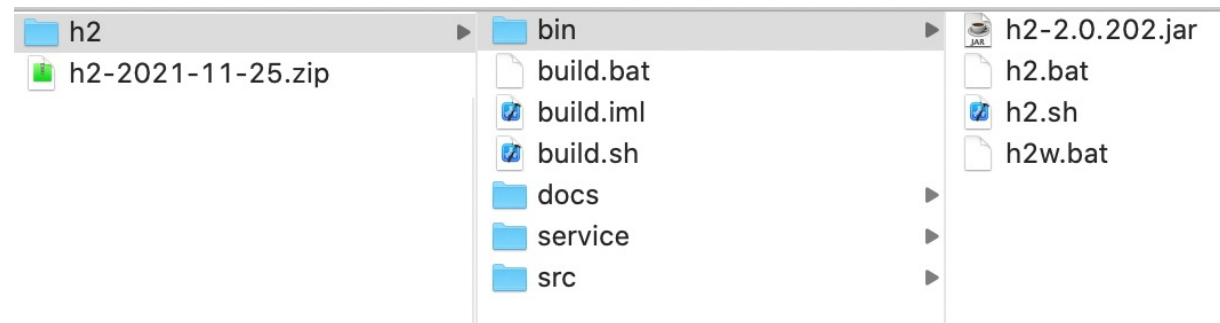
[Archive Downloads](#)

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`

← → ↻ ⓘ localhost:8082/test.do?jsessionid=cbca28ec8e8ae3b02d1c4baa302f7d7f

English ▼ Preferences Tools Help

Login

Saved Settings: Generic H2 (Server) ▼

Setting Name: Generic H2 (Server) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:tcp://localhost/~ /Download/h2test

User Name: sa

Password: ••

Connect Test Connection

Test successful

Access H2 with browser

← → ↻ ⓘ localhost:8082/login.do?jsessionid=cbca28ec8e8ae3b02d1c4baa302f7d7f

🎵 | 🧩 | ☒ Auto commit 🧩 | 🖨️ | Max rows: 1000 ▼ | 🟢 | 📄 | Auto complete Off ▼ Auto select On ▼ ⓘ

📁 jdbc:h2:tcp://localhost/~/Download Run Run Selected Auto complete Clear SQL statement:

+ 📁 INFORMATION_SCHEMA

+ 👤 Users

📘 H2 2.0.202 (2021-11-25)

```
CREATE TABLE TEST(ID INT PRIMARY KEY,
NAME VARCHAR(255));
```

Important Commands

📘		Displays this Help Page
📄		Shows the Command History
🟢	Ctrl+Enter	Executes the current SQL statement
🟢	Shift+Enter	Executes the SQL statement defined by the text selection
	Ctrl+Space	Auto complete
🔌		Disconnects from the database

Sample SQL Script

Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;
Change data in a row	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Remove a row	DELETE FROM TEST WHERE ID=2;
Help	HELP ...

Access H2 with browser

ALSO:

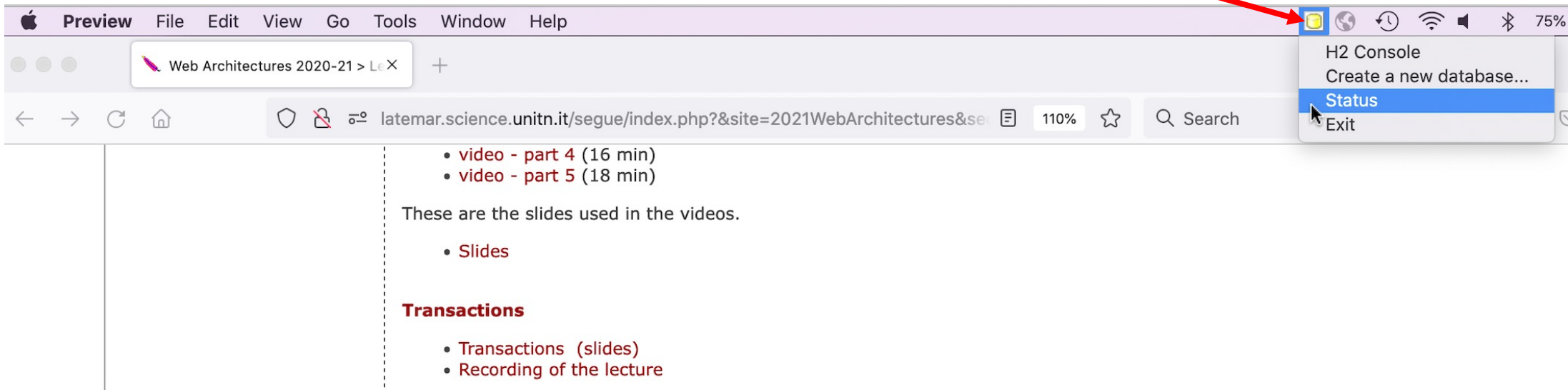


Table definition

```
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 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>
  </datasources>
  <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>
</subsystem>
```

```

<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

standalone.xml

you must change the content of the datasources section like this

```

<drivers>
  <driver name="h2" module="org.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>

```

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

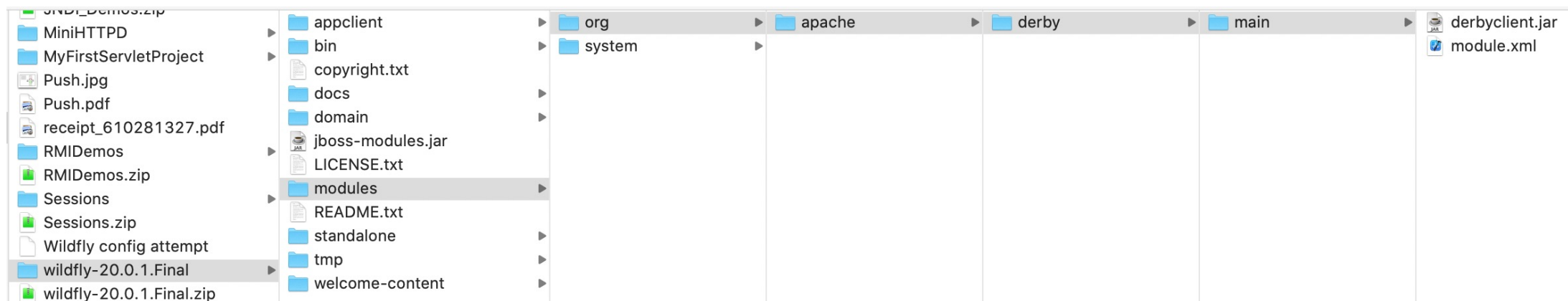
Derby example – not working!

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

← → ↻ ⓘ localhost:9990/console/index.html#configuration;path=configuration~subsystems!css~datasou

HAL Management Console

Homepage Deployments **Configuration** Runtime Patching Access Control

Configuration	Subsystem (32)	Datasources & Drivers	JDBC Driver  
Subsystems >	<i>Filter by: name or subtitle</i>	Datasources >	<i>Filter by: driver name or provide</i>
Interfaces >	Batch	JDBC Drivers >	 h2
Socket Bindings >	JBeret		 org.apache.derby
Paths	Core Management		
System Properties	Datasources & Drivers >		
	Deployment Scanners		
	Discovery		
	Distributable Web		

Connect to the console, check the datasource

← → ↻ ⓘ localhost:9990/console/index.html#configuration;path=configuration~subsystems!css~datasources!data-source-... ☆ S [copy] [gear] M

HAL Management Console

🔔 admin1

HomepageDeployments**Configuration**RuntimePatchingAccess Control

Configuration	Subsystem (32)	Datasources & Drivers	Datasource <div>⊕ ▾ ↺</div>
Subsystems >	<i>Filter by: name or subtitle</i>	Datasources >	<i>Filter by: name, xa, .../disabled, c</i>
Interfaces >	Batch		✔ Derby... <div>View ▾</div>
Socket Bindings >	JBeret	JDBC Drivers >	✔ ExampleDS
Paths	Core Management		
System Properties	Datasources & Drivers >		
	Deployment Scanners		
	Discovery		

DerbyDS

Datasource

✔ The datasource **DerbyDS** is enabled. [Disable](#)

Main Attributes

JNDI Name:	java:jboss/datasources/DerbyDS
Driver Name:	org.apache.derby
Connection URL:	jdbc:derby://localhost:1527/SESSIO...



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

```
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();
    }
}
```

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

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

@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;

}

```
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;
```

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

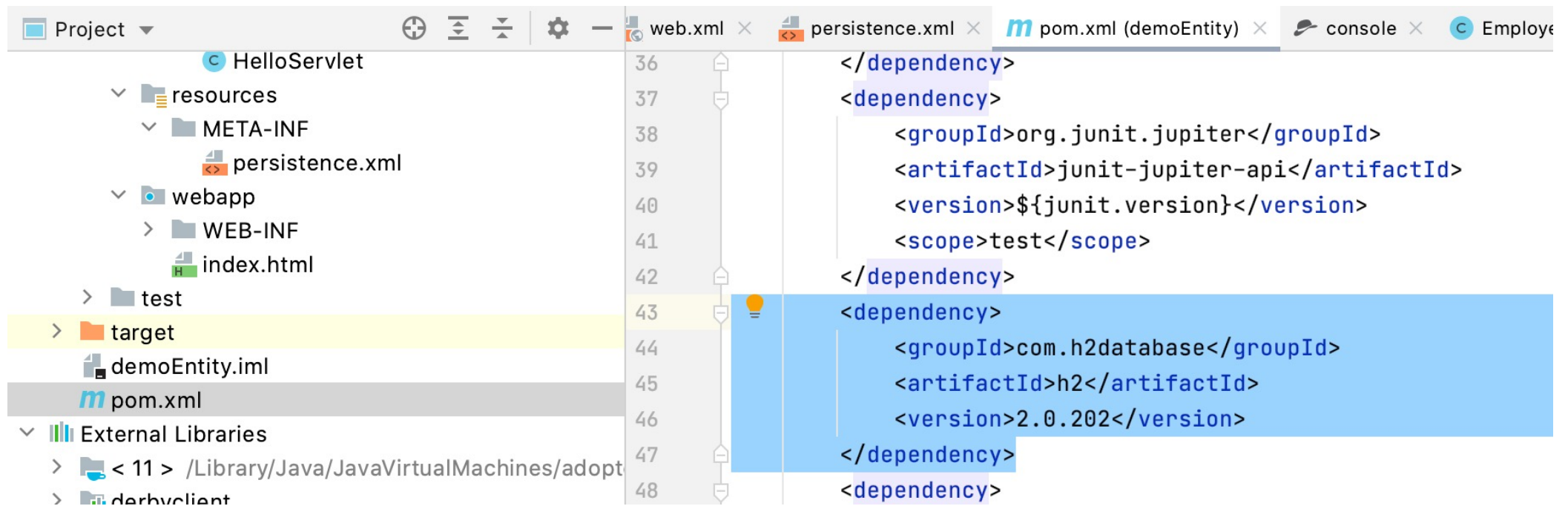
In the standalone.xml, add the following section (in the datasource section)

```
<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 screenshot displays the IntelliJ IDEA interface for a JEE project. The Project tool window on the left shows the project structure, including the 'pom.xml' file. The main editor shows the 'pom.xml' file with the following dependencies:

```
36 </dependency>
37
38 <dependency>
39     <groupId>org.junit.jupiter</groupId>
40     <artifactId>junit-jupiter-api</artifactId>
41     <version>${junit.version}</version>
42     <scope>test</scope>
43 </dependency>
44
45 <dependency>
46     <groupId>com.h2database</groupId>
47     <artifactId>h2</artifactId>
48     <version>2.0.202</version>
49 </dependency>
```

The 'h2database' dependency is highlighted in blue, indicating it is the current focus.

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;

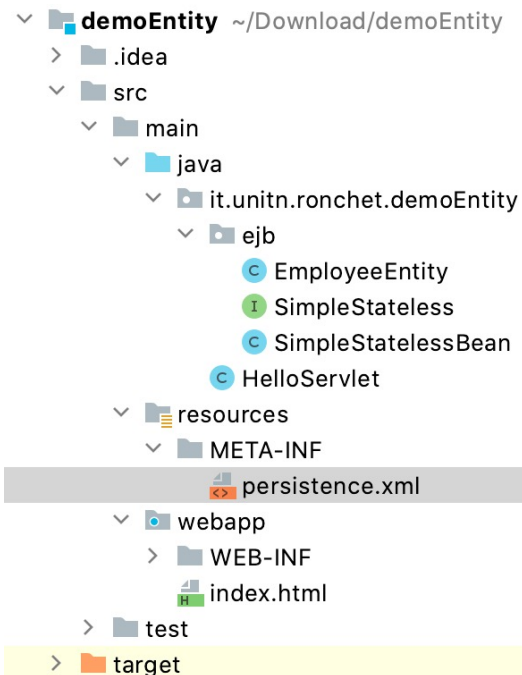
    EmployeeEntity that = (EmployeeEntity) o;

    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



```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5                                 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
6             version="2.2">
7     <persistence-unit name="default">
8         <jta-data-source>java:jboss/datasources/TestDS</jta-data-source>
9     </persistence-unit>
10 </persistence>
11
```

The remote interface

included both in the client and in the server!

The screenshot shows an IDE window with the following components:

- Project Explorer (Left):** Displays the project structure for `demoEntity`. The path `src/main/java/it/unitn/ronchet/demoEntity/ejb` is expanded, showing files `EmployeeEntity`, `SimpleStateless` (highlighted), `SimpleStatelessBean`, and `HelloServlet`. Other folders like `resources`, `webapp`, and `test` are also visible.
- Editor (Right):** Shows the code for `SimpleStateless`. The code is as follows:

```
1 package it.unitn.ronchet.demoEntity.ejb;
2
3 public interface SimpleStateless {
4     String getValue(int i);
5 }
6
```

The stateless bean (in the EJB server)

The screenshot displays an IDE interface with the following components:

- Project Structure (Left Panel):** Shows the project hierarchy for `demoEntity`. The `src/main/java/it/unitn/ronchet/demoEntity/ejb` package is expanded, highlighting `SimpleStatelessBean`. Other files visible include `EmployeeEntity`, `SimpleStateless`, `SimpleStatelessBean`, `HelloServlet`, `resources/META-INF/persistence.xml`, `webapp/WEB-INF/index.html`, `target/demoEntity.iml`, `pom.xml`, `External Libraries`, and `Scratches and Consoles`.
- Code Editor (Right Panel):** Displays the source code for `SimpleStatelessBean.java`. The code is as follows:

```
1 package it.unitn.ronchet.demoEntity.ejb;
2
3 import javax.ejb.Remote;
4 import javax.ejb.Stateless;
5 import javax.persistence.EntityManager;
6 import javax.persistence.PersistenceContext;
7 import javax.persistence.Query;
8
9 @Stateless
10 @Remote(SimpleStateless.class)
11 public class SimpleStatelessBean implements SimpleStateless {
12     @PersistenceContext(unitName="default")
13     private EntityManager entityManager;
14
15     @Override
16     public String getValue(int i) {
17         Query q=entityManager.createQuery( s: "From EmployeeEntity where ID = "+i);
18         EmployeeEntity e=(EmployeeEntity)(q.getSingleResult());
19         return e.getLastname();
20     }
21 }
```


The servlet (in the Web server)

The screenshot displays an IDE interface with a project named "demoEntity". The left sidebar shows the project structure, including folders like ".idea", "src", "main", "java", "it.unitn.ronchet.demoEntity", "ejb", "resources", "META-INF", "webapp", "WEB-INF", "test", "target", and files like "demoEntity.iml", "pom.xml", "persistence.xml", "index.html". The right pane shows the code for "HelloServlet.java", which imports "javax.servlet.*" and implements the "doGet" method to return "Hello World!".

```
import javax.servlet.*;

@WebServlet(name = "helloServlet", value = "/hello-servlet")
public class HelloServlet extends HttpServlet {
    private String message;

    public void init() { message = "Hello World!"; }

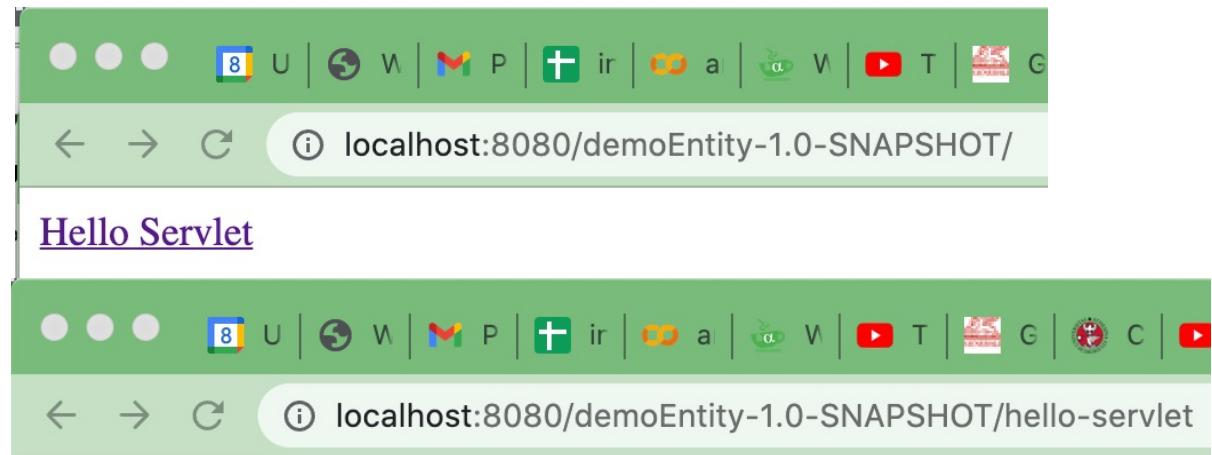
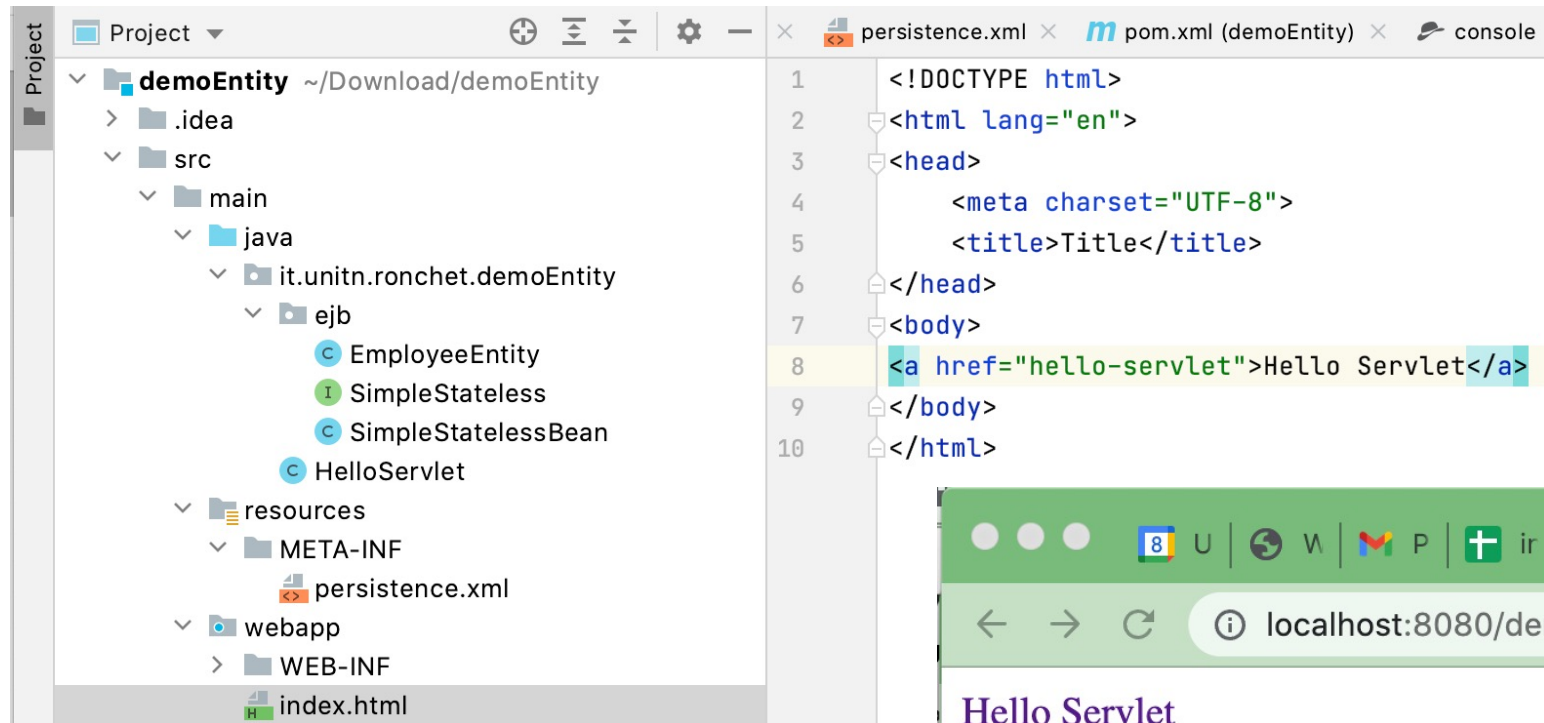
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        response.setContentType("text/html");

        // Hello
        Context ctx = null;
        SimpleStateless hello=null;
        try {
            ctx = new InitialContext();
            String name="java:module/SimpleStatelessBean!it.unitn.ronchet.demoEntity.ejb.SimpleStateless";
            hello= (SimpleStateless) ctx.lookup(name);
        } catch (NamingException e) {
            e.printStackTrace();
        }

        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h1>" + hello.getValue( 1 ) + "</h1>");
        out.println("</body></html>");
    }

    public void destroy() {
    }
}
```

index.html (in the Web server)



Rossi

On the console, let's check the deployment

← → ↻ ⓘ localhost:9990/console/index.html#deployment;deployment=demoEntity-1.0-SNAPSHOT

HAL Management Console

« Back / Deployment ⇒ demoEntit...SNAPSHOT ▾

Management Model

↻ -

demoEntity-1.0-SNAPSHOT

- subdeployment
- subsystem
 - ejb3
 - message-driven-bean
 - singleton-bean
 - stateful-session-bean
 - stateless-session-bean
 - SimpleStatelessBean
 - jpa
 - hibernate-persistence-unit
 - demoEntity-1.0-SNAPSHOT#default
 - logging
 - microprofile-opentracing-smallrye
 - undertow
 - batch-jberet
 - datasources

A deployment represents anything that can be deployed (e.g. an application RAR or JBoss-specific deployment) into a server.

Data Attributes Operations

Edit Reset Help

Disabled Time	MILLISECONDS
Disabled Timestamp	
Enabled	true
Enabled Time	1638990695960 MILLISECONDS
Enabled Timestamp	2021-12-08 20:11:35,960 CET
Managed	false
Name	demoEntity-1.0-SNAPSHOT
Persistent	true

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) HV0000001: 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

The screenshot displays an IDE interface with a Maven project named `demoEntity`. The `pom.xml` file is open, showing the following dependencies:

```
<dependency>
<groupId>org.junit.jupiter</groupId>
<artifactId>junit-jupiter-api</artifactId>
<version>${junit.version}</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
<version>2.0.202</version>
</dependency>
<dependency>
<groupId>org.junit.jupiter</groupId>
```

The console output shows the following log messages:

```
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 bean

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
11:38,070 INFO [io.iaegertracing.internal.JaegerTracer] (MSC service thread 1-8) No shutdown hook registered: Plea
```

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):

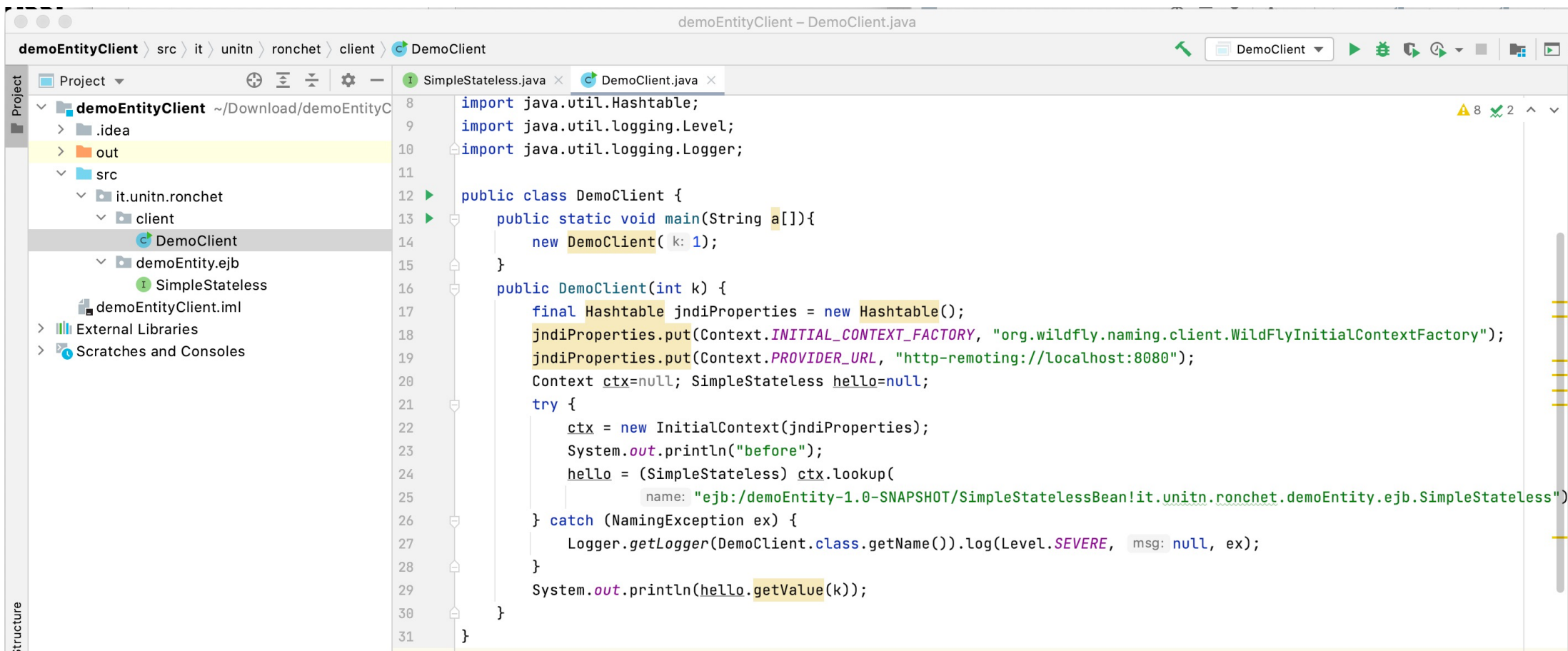
- copy in it the Remote interface
- write the client code (using the right JNDI bindings):

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

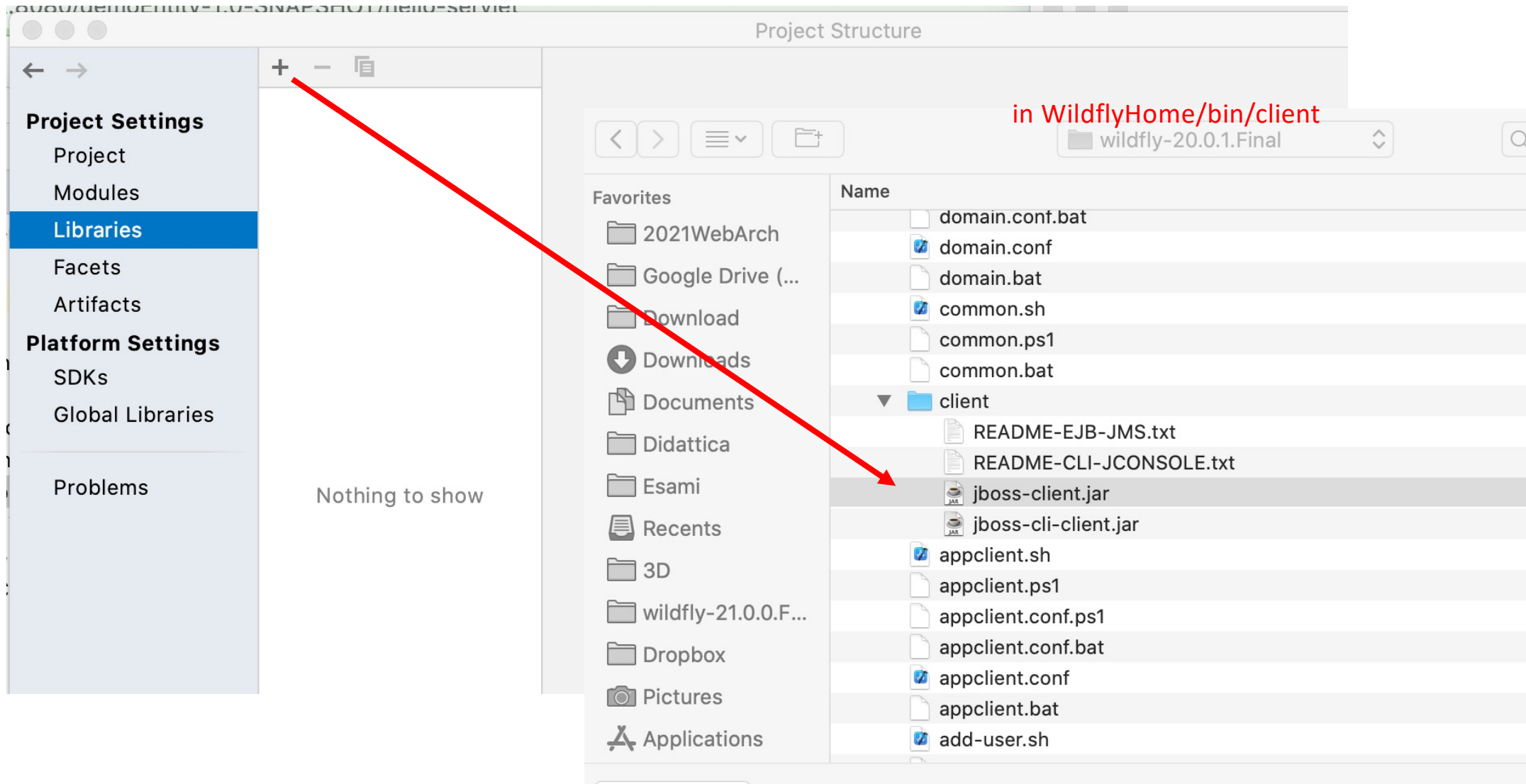
jndiProperties.put(Context.PROVIDER_URL,
    "http-remoting://localhost:8080");

try {ctx = new InitialContext(jndiProperties);}
catch (NamingException ex) {ex.printStackTrace();}
```

Let's create the client project

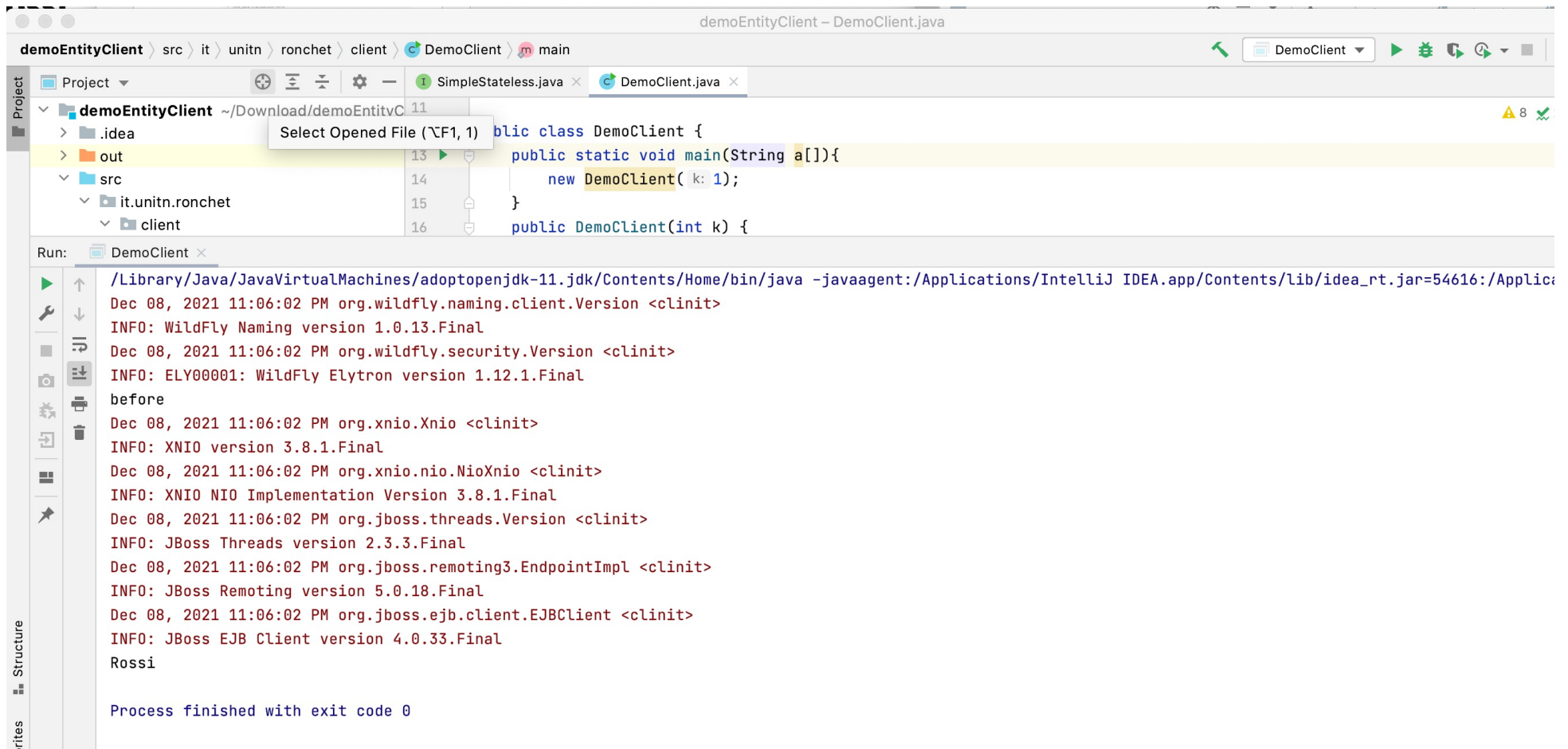


Add libraries to the client project



Now we can run the client project

Of course, the server part must be deployed first.



The screenshot shows the IntelliJ IDEA IDE with the `demoEntityClient` project open. The `src` directory contains the `it` package, which in turn contains the `unitn` package, `ronchet` package, and `client` package. The `client` package contains the `DemoClient.java` file. The `DemoClient.java` file is open in the editor, showing the following code:

```
public class DemoClient {  
    public static void main(String a[]){  
        new DemoClient( k: 1);  
    }  
    public DemoClient(int k) {  
        // ...  
    }  
}
```

The `Run` button is clicked, and the output console shows the following log:

```
/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=54616:/Applic  
Dec 08, 2021 11:06:02 PM org.wildfly.naming.client.Version <clinit>  
INFO: WildFly Naming version 1.0.13.Final  
Dec 08, 2021 11:06:02 PM org.wildfly.security.Version <clinit>  
INFO: ELY00001: WildFly Elytron version 1.12.1.Final  
before  
Dec 08, 2021 11:06:02 PM org.xnio.Xnio <clinit>  
INFO: XNIO version 3.8.1.Final  
Dec 08, 2021 11:06:02 PM org.xnio.nio.NioXnio <clinit>  
INFO: XNIO NIO Implementation Version 3.8.1.Final  
Dec 08, 2021 11:06:02 PM org.jboss.threads.Version <clinit>  
INFO: JBoss Threads version 2.3.3.Final  
Dec 08, 2021 11:06:02 PM org.jboss.remoting3.EndpointImpl <clinit>  
INFO: JBoss Remoting version 5.0.18.Final  
Dec 08, 2021 11:06:02 PM org.jboss.ejb.client.EJBClient <clinit>  
INFO: JBoss EJB Client version 4.0.33.Final  
Rossi  
  
Process finished with exit code 0
```



Simple working example
with Detached Entity as DTO

Modified server project

The screenshot displays an IDE interface for a project named **demoEntity**. The breadcrumb navigation at the top indicates the current file path: **demoEntity > src > main > java > it > unitn > ronchet > demoEntity > ejb > SimpleStateless**. The left sidebar shows the project structure, with the **SimpleStateless** interface selected. The main editor area shows the code for **SimpleStateless.java**, which is as follows:

```
1 package it.unitn.ronchet.demoEntity.ejb;
2
3 public interface SimpleStateless {
4     String getValue(int i);
5     EmployeeEntity getEmployee(int i);
6 }
7
```

The line `EmployeeEntity getEmployee(int i);` is highlighted with a red rectangular box.

Modified server project

The screenshot displays an IDE interface for a project named 'demoEntity'. The left sidebar shows the project structure, including the 'src/main/java/it/unitn/ronchet/demoEntity/ejb' package, which contains 'EmployeeEntity', 'SimpleStateless', and 'SimpleStatelessBean'. The 'SimpleStatelessBean' class is selected. The main editor window shows the code for 'SimpleStatelessBean.java', which implements the 'SimpleStateless' interface. The code includes imports for 'javax.ejb.Remote', 'javax.ejb.Stateless', 'javax.persistence.EntityManager', 'javax.persistence.PersistenceContext', and 'javax.persistence.Query'. The class is annotated with '@Stateless' and '@Remote(SimpleStateless.class)'. It has a private 'EntityManager entityManager' field. The 'getValue(int i)' method is annotated with '@Override' and returns the last name of an 'EmployeeEntity' found by ID. The 'getEmployee(int i)' method is also annotated with '@Override' and returns the 'EmployeeEntity' found by ID. The 'getEmployee' method is highlighted with a red rectangle.

```
3 import javax.ejb.Remote;
4 import javax.ejb.Stateless;
5 import javax.persistence.EntityManager;
6 import javax.persistence.PersistenceContext;
7 import javax.persistence.Query;
8
9 @Stateless
10 @Remote(SimpleStateless.class)
11 public class SimpleStatelessBean implements SimpleStateless {
12     @PersistenceContext(unitName="default")
13     private EntityManager entityManager;
14
15     @Override
16     public String getValue(int i) {
17         Query q=entityManager.createQuery( s: "From EmployeeEntity where ID = "+i);
18         EmployeeEntity e=(EmployeeEntity)(q.getSingleResult());
19         return e.getLastname();
20     }
21
22     @Override
23     public EmployeeEntity getEmployee(int i) {
24         Query q=entityManager.createQuery( s: "From EmployeeEntity where ID = "+i);
25         EmployeeEntity e=(EmployeeEntity)(q.getSingleResult());
26         return e;
27     }
28 }
```

Modified client project

demoEntityClient > src > it > unitn > ronchet > client > DemoClient

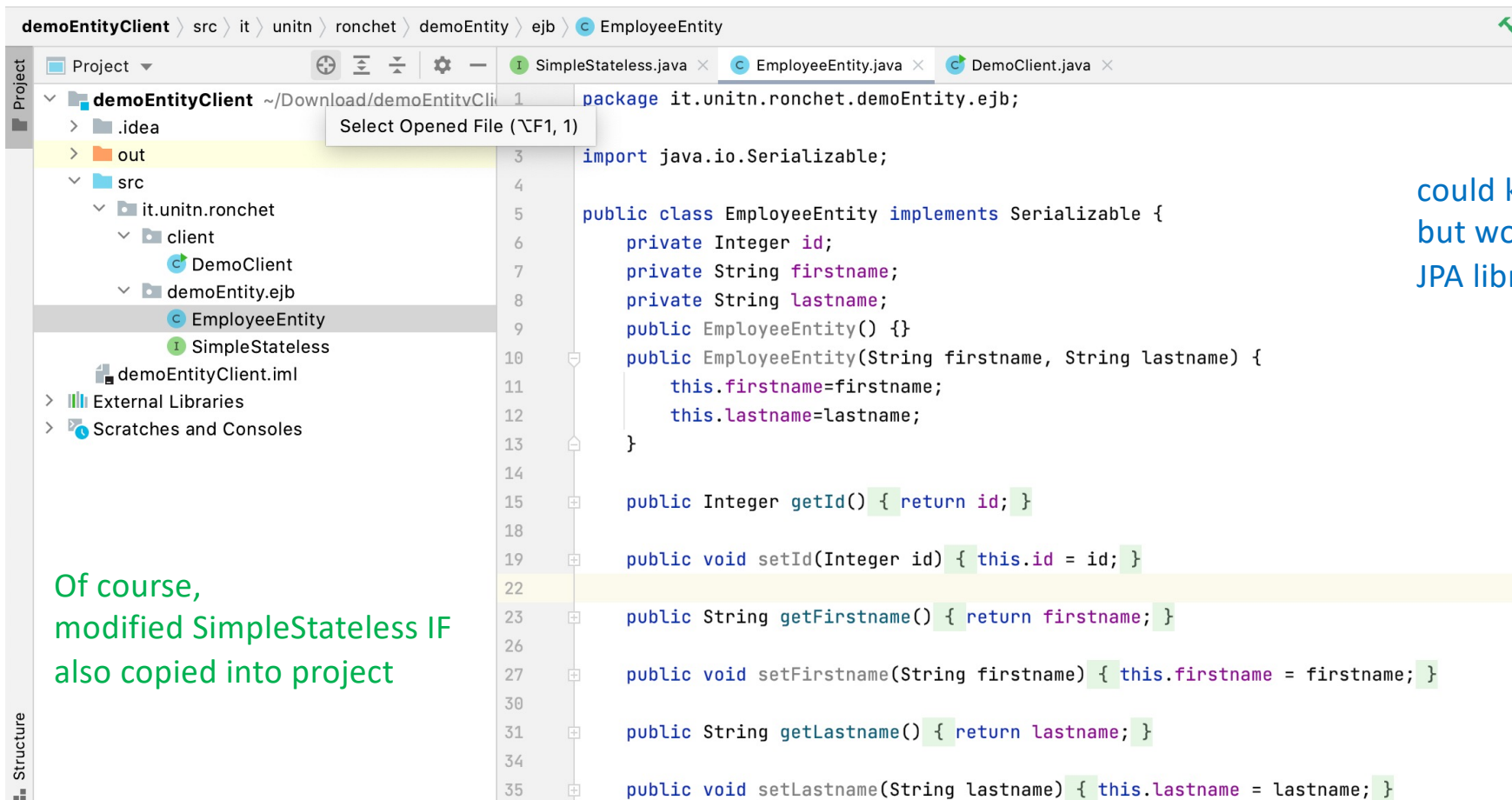
Project

- demoEntityClient ~/Download/demoEntityCli
- > .idea
- > out
- > src
 - it.unitn.ronchet
 - client
 - DemoClient
 - demoEntity.ejb
 - EmployeeEntity
 - SimpleStateless
 - demoEntityClient.iml
 - External Libraries
 - Scratches and Consoles

```
12 public class DemoClient {
13
14     public static void main(String a[]){
15         new DemoClient(1);
16     }
17
18     public DemoClient(int k) {
19         final Hashtable jndiProperties = new Hashtable();
20         jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY, "org.wildfly.naming.client.WildFlyInitialContextFactory");
21         jndiProperties.put(Context.PROVIDER_URL, "http-remoting://localhost:8080");
22         Context ctx=null; SimpleStateless hello=null;
23         try {
24             ctx = new InitialContext(jndiProperties);
25             System.out.println("before");
26             hello = (SimpleStateless) ctx.lookup(
27                 "ejb:/demoEntity-1.0-SNAPSHOT/SimpleStatelessBean!it.unitn.ronchet.demoEntity.ejb.SimpleStateless");
28         } catch (NamingException ex) {
29             Logger.getLogger(DemoClient.class.getName()).log(Level.SEVERE, null, ex);
30         }
31         System.out.println(hello.getValue(k));
32         EmployeeEntity e=hello.getEmployee(k);
33         System.out.println(e.getId()+" "+e.getFirstname()+" "+e.getLastname());
34     }
35 }
```

Modified client project

copied without annotations



The screenshot shows an IDE interface with a project structure on the left and a code editor in the center. The project structure is as follows:

- demoEntityClient
 - .idea
 - out
 - src
 - it.unitt.ronchet
 - client
 - DemoClient
 - demoEntity.ejb
 - EmployeeEntity
 - SimpleStateless
 - demoEntityClient.iml
 - External Libraries
 - Scratches and Consoles

The code editor shows the following Java code for `EmployeeEntity.java`:

```
1 package it.unitt.ronchet.demoEntity.ejb;
2
3 import java.io.Serializable;
4
5 public class EmployeeEntity implements Serializable {
6     private Integer id;
7     private String firstname;
8     private String lastname;
9     public EmployeeEntity() {}
10    public EmployeeEntity(String firstname, String lastname) {
11        this.firstname=firstname;
12        this.lastname=lastname;
13    }
14
15    public Integer getId() { return id; }
16
17    public void setId(Integer id) { this.id = id; }
18
19    public String getFirstname() { return firstname; }
20
21    public void setFirstname(String firstname) { this.firstname = firstname; }
22
23    public String getLastname() { return lastname; }
24
25    public void setLastname(String lastname) { this.lastname = lastname; }
```

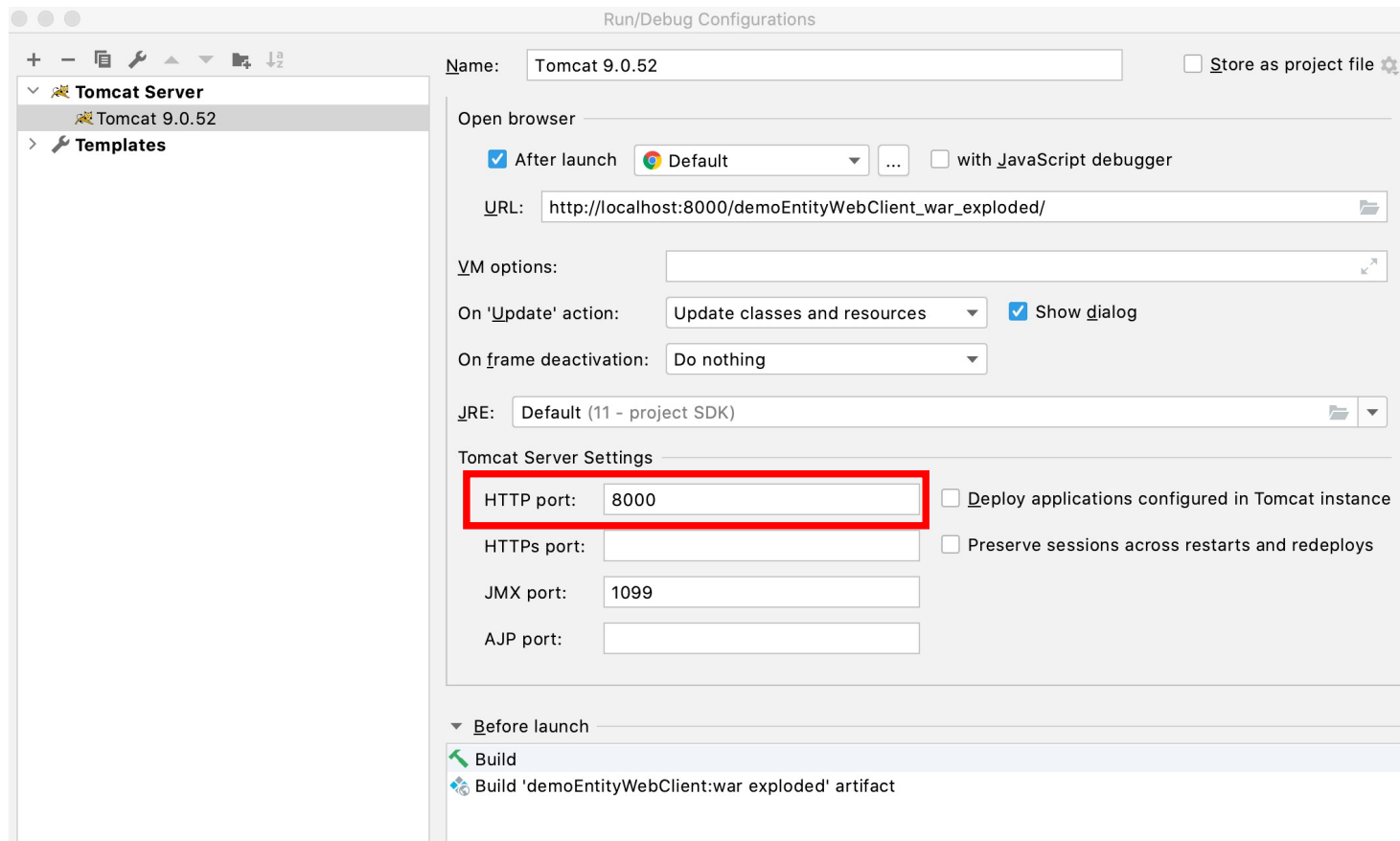
Of course,
modified SimpleStateless IF
also copied into project

could keep annotations,
but would need to include
JPA libraries



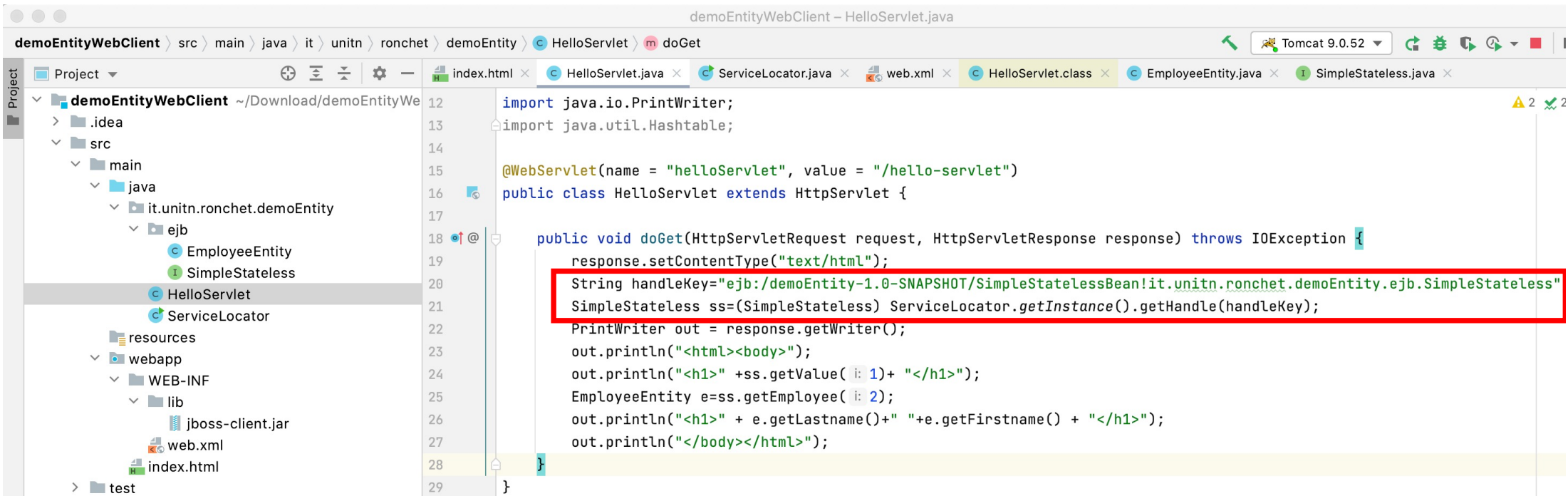
Simple working example
with External Web Server

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



Service Locator – part 1

The screenshot displays an IDE interface with a project structure on the left and a code editor on the right. The project structure shows a web application named `demoEntityWebClient` with a `src/main/java` directory containing the `it.unitn.ronchet.demoEntity` package. The `ServiceLocator` class is highlighted in the package list. The code editor shows the implementation of `ServiceLocator`, which is a static class that manages a map of service instances. It includes imports for `SimpleStateless`, `WildFlyInitialContextFactory`, `Context`, `InitialContext`, `NamingException`, `HashMap`, and `Hashtable`. The `getInstance()` method is synchronized and creates a new `ServiceLocator` instance if the static `serviceLocator` is null. The `private ServiceLocator()` constructor calls `getContext()` and initializes the `map` with a new `HashMap`.

```
1 package it.unitn.ronchet.demoEntity;
2 import it.unitn.ronchet.demoEntity.ejb.SimpleStateless;
3 import org.wildfly.naming.client.WildFlyInitialContextFactory;
4
5 import javax.naming.Context;
6 import javax.naming.InitialContext;
7 import javax.naming.NamingException;
8 import java.util.HashMap;
9 import java.util.Hashtable;
10
11 public class ServiceLocator {
12     private static ServiceLocator serviceLocator=null;
13     private Context ctx=null;
14     private HashMap<String,Object> map;
15
16     public static synchronized ServiceLocator getInstance(){
17         if (serviceLocator == null) {
18             serviceLocator = new ServiceLocator(); }
19         return serviceLocator;
20     }
21     private ServiceLocator(){
22         getContext();
23         map=new HashMap<String,Object>();
24     }
```

Service Locator – part 2

The screenshot displays an IDE interface with two main panels. The left panel shows the project's file structure:

- Project**: demoEntityWebClient ~ /Download/demoEntityWe...
 - .idea
 - src
 - main
 - java
 - it.unitm.ronchet.demoEntity
 - ejb
 - EmployeeEntity
 - SimpleStateless
 - HelloServlet
 - ServiceLocator
 - resources
 - webapp
 - WEB-INF
 - lib
 - jboss-client.jar
 - web.xml
 - index.html
 - test
 - target
 - classes
 - it
 - unitm
 - ronchet
 - demoEntity
 - ejb
 - HelloServlet
 - ServiceLocator
 - demoEntityWebClient-1.0-SNAPSHOT
 - generated-sources
 - annotations
 - demoEntityWebClient.iml
 - pom.xml
 - External Libraries

The right panel shows the source code of the `ServiceLocator.java` file:

```

private Context getContext() {
    if (ctx==null) {
        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();
        }
    }
    return ctx;
}

Object getHandle(String s) {
    Object retval=null;
    if (! map.containsKey(s)) {
        try {
            retval=ctx.lookup(s);
            map.put(s,retval);
        } catch (NamingException e) {
            e.printStackTrace();
        }
    } else {
        retval=map.get(s);
    }
    return retval;
}

public static void main(String a[]){
    String handleKey="ejb:/demoEntity-1.0-SNAPSHOT/SimpleStatelessBean!it.unitm.ronchet.demoEntity.ejb.SimpleStateless";
    SimpleStateless ss=(SimpleStateless) ServiceLocator.getInstance().getHandle(handleKey);
    System.out.println(ss.getValue( : 2));
}

```