# Exercise 2 Implementing the Shop with EJB

## 2.1 Overview

This exercise is a hands-on exercise in Enterprise JavaBeans (EJB). The exercise is as similar as possible to the other exercises (in other technologies). We will implement a database connected fruit shop with a simple web access point.

### 2.1.1 Goals

After completing this exercise you should be able to:

- Give an overview of the EJB component technology
- Use entity classes and the Java Persistence API to maintain persistent data.
- Use session beans to maintain session specific data.
- Create web clients with Java Servlets.
- Compare the amount and type of work needed to accomplish the same task in the different technologies discussed during the course.

### 2.1.2 System Requirements

See online tutorial on Java EE at http://netbeans.org/kb/docs/javaee/javaee-gettingstarted.html.

### 2.1.3 Prerequisites

You should be familiar with Java (or some similar object oriented language) and have read the EJB chapter in the book. Use the lecture slides and (not or) tutorials etc. available at http://java.sun.com and http://netbeans.org, in particular the abovementioned tutorial on Java EE.

### 2.1.4 Examination

You should be able to explain all parts of your produced software.

The application should be zipped and submitted in Blackboard (see http://mdh.blackboard.com for instructions).

You are allowed to be at a maximum two persons in each group. Both must know – and be able to answer questions about – all parts of the produced software.

## 2.1.5 Application Overview

The idea is to implement a fruit shop, similar to the other exercises. The different kinds of fruits are stored in a database. To represent an entity in this database, we use an entity class (`Fruits`). One instance represents one row in the `FRUITS` database table. We will use the Java Persistence API for mapping the rows in the database to entity class instances. (In EJB 2.0, *container managed persistence* was the preferred way to link classes and databases, but Sun has now declared this method deprecated.)

The `Fruits` class has only local interfaces since it is never accessed directly from a remote user. Instead we use a "façade" to the shop. This is a stateless session bean (`FruitsFacade`). The idea is that `Fruits` has a local interface for fine grained access and `FruitsFacade` has a course grained interface, suitable for remote communication. To keep track of each customer's choices, we can implement a stateful session bean (`ShoppingCart`). The shopping cart is a session bean since it does not represent an entity in a database, but something that should exist during a session only. Each session will have its own instance (meaning that all customers will have their own shopping carts).

The user is presented with a list of fruits available to buy on a web page. This page is produced by a "servlet" component on the server side. It is up to you to implement a full scale shop, in the exercise we will use the servlet to test the beans only. The relationship between the different parts of the application is described in the following picture.

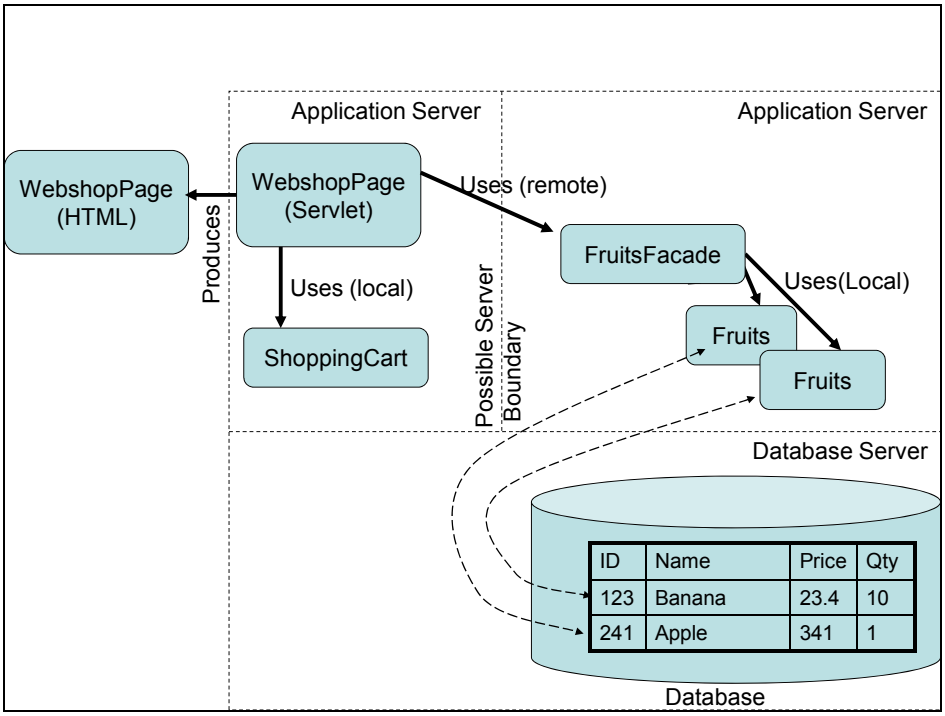


*Figure 2.1 Web shop artifact relationship overview*

## 2.2 Setting up a private development environment

This exercise uses the GlassFish v3 application server and the Java DB database server from Sun. If you wish to use your own Linux or Windows machine instead of the MDH equipment, you can download the Java EE 6 SDK and the NetBeans IDE 6.8 from http://java.sun.com/javaee/downloads/. All software used in this exercise is available for free.

The rest of this document assumes you will be working in the computer rooms and use "H:\cdt401" as your working directory. Since we are several groups sharing the same machines and want to work without interfering with each other, we have to create our own domains and databases.

1. Start the NetBeans IDE 6.8 and select the *Services* tab.
2. Create a personal domain:
   a. Right-click *Servers* and select *Add Server…*
   b. Select *GlassFish v3* and type a server name, e.g. "MyServer".
   c. Click *Next* twice to go to the *Domain Name/Location* page.
   d. Select *Register Local Domain* and type the path to where you wish to create the domain, e.g. "H:\cdt401\domains\MyDomain". (Folders will be automatically created if necessary.)
   e. Click *Finish* and wait (creating the domain may take some time).
   f. Expand *Servers* to verify that your server has been added. Right-click the server to start it (this may take some time as well). Right-click it once more to stop it again.
3. Create the database for the fruits:
   a. Expand *Databases*, right-click *Java DB*, and select *Create Database…*
   b. Type "FruitShop" as the database name and select a username and password (which you will need to remember). The rest of this document assumes the username is "cdt401".
   c. Click the *Properties…* button and change the *Database Location* to where you wish to store the database, e.g. "H:\cdt401\.netbeans-derby".
   d. Click *OK* twice and wait while the database is created.
   e. Right-click the connection *jdbc:derby://localhost:1527/FruitShop [cdt401 on CDT401]* and select *Connect…*
   f. Right-click the connection again and select *Execute Command…*
   g. Type or copy/paste the below SQL commands into the window that opens and click the *Run SQL* button (in the same window, directly to the right of the *Connection* field).

```
CREATE TABLE Fruits
(
  id VARCHAR(8) CONSTRAINT pk_Fruit PRIMARY KEY,
  name VARCHAR(24),
  price DOUBLE PRECISION,
  quantity INTEGER
);
```

h. Expand the connection down to *CDT401/Tables/FRUITS*, right-click *FRUITS*, and select *View Data…* (if you do not see CDT401, right-click the connection and select *Refresh*).

i. Click the *Insert Record(s)* button (or press Alt + I) and enter some data (just like you did in Exercise 1).

If you have chosen to use to the paths and names suggested above, you should now have a personal domain at “H:\cdt401\domains\MyDomain” and a database at “H:\cdt401\.netbeans-derby\FruitShop”.

## 2.3 Working Steps

The exercise will start with creating the `Fruits` entity class that represents one row in the `FRUITS` database table. We will then create the `FruitsFacade` session been and a simple servlet that (possibly remotely) fetches some contents from the database. These will be deployed and tested before creating the shopping cart session bean and the complete web shop (optional exercises). We will use the NetBeans IDE and its wizards to create the code, but let us first take a look at what happens behind the scenes.

### 2.3.1 Persistence Overview

As already mentioned, the `Fruits` entity class should represent a row in the database and we will use the Java Persistence API to handle the database mapping. Instead of writing the code for the class, we use a NetBeans wizard to generate it from the existing database. For each property we wish to persist, the class has simple set/get methods. The generated file `Fruits.java` will furthermore contain a number of annotations defining named queries, which can be used to fetch data from the database in the form of `Fruits` objects. In addition to this file, the wizard also generates a number of resources: a persistence unit (PU), a data source (or JDBC resource) and a database connection pool. These are described further below.

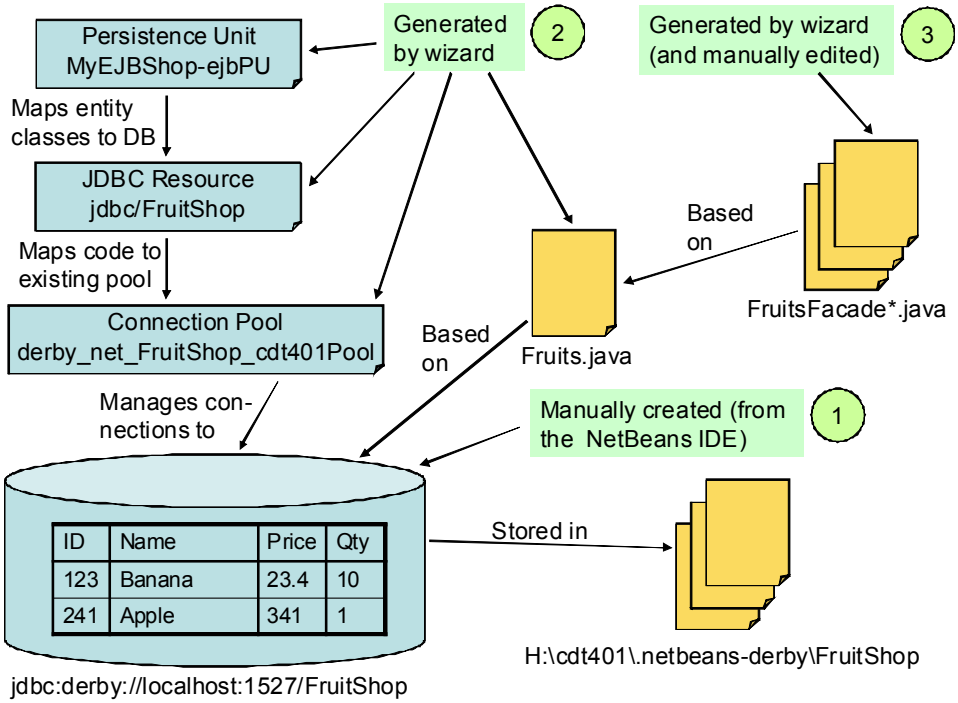


*Figure 2.2 Details of connection to the database.*

The persistence setup is described in the figure above. The data is physically stored in a folder on disk, but since the database server could be run on another machine, and for efficiency reasons, you have to use a connection pool that manages all connections to the database. This also gives you the opportunity to change from Java DB to another database, such as MySQL or Oracle, without changing your code. During deployment, the JDBC resource is used to identify which pool to use. The Java code references the JDBC resource indirectly through the PU, which is defined in a file called `persistence.xml` (not shown). Note that the names generated by NetBeans may differ from those shown here.

To access the data from the user interface, which could potentially be running on a separate web server, we also need the `FruitsFacade` session bean. In this exercise, you will first use another NetBeans wizard to generate both interfaces and an implementation class for the bean, and then edit the generated files to achieve the desired functionality. The two interfaces are `FruitsFacadeLocal` with fine-grained access methods and `FruitsFacadeRemote` with course-grained access methods. The last step of the exercise is to implement a simple servlet to test both these interfaces (not shown in the figure).

## 2.3.2 Creating the Entity Class with the NetBeans IDE

1. Open the tutorial at http://netbeans.org/kb/docs/javaee/javaee-gettingstarted.html. Read the tutorial together with the instructions here to get the coding details.
2. Create a new project in the NetBeans IDE:
   a. Click the *New Project* button and select *Enterprise Application* from the *Java EE* category.
   b. Name the project "MyEJBShop" and change the *Project Folder* to where you wish to store your files, e.g. "H:\cdt401\NetBeansProjects".
   c. Select the server you defined in Section 2.2 and the version *Java EE 6*. Make sure that *Create EJB Module* and *Create Web Application Module* are both checked while *Create Client Application Module* is unchecked.
   d. Click *Finish* and wait for the project to be created
3. Select the *Projects* tab and create the entity class:
   a. Right-click *MyEJBShop-ejb* and select *New* | *Entity Classes from Database*…
   b. Select *New Data Source*… from the *Data Source* drop-down list.
   c. Type "jdbc/FruitShop" as the *JNDI Name* and, select the connection *jdbc:derby://localhost:1527/FruitShop [cdt401 on CDT401]*, and click *OK*.
   d. Add *FRUITS* to the *Selected Tables* list and click *Next*.
   e. Type "entities" as the *Package* and make sure that *Generate Named Query Annotations for Persistent Fields* is checked.
   f. Click on the *Create Persistence Unit* button.
   g. Select *jdbc/FruitShop* as the *Data Source* and *None* as the *Table Generation Strategy*.
   h. Click *Create* and then *Finish*.
4. Expand *MyEJBShop-ejb/Source Packages/entities* and open the generated file `Fruits.java`. Carefully read through the file and make sure that you understand the code that was generated. Compare with the tutorial, which creates an entity class without an existing database.

## 2.3.3 Implementing the Session Bean

1. Follow the steps under *Creating the Session Facade* in the tutorial to create a session bean for the *Fruits* entity class, but select to create both a local and a remote interface. Expand *MyEJBShop-ejb/Source Packages/boundary* to see the three generated files.
2. Open `FruitsFacadeLocal.java` and add one method to the interface:

```
Fruits findByName(String name);
```

3. Open `FruitsFacadeRemote.java` and replace all the methods in the interface with one single method, as shown below. Hint: after editing the code, right-click in the window and select *Fix Imports*.

```
public interface FruitsFacadeRemote {
    Collection getFruitsList();
}
```

4. Open `FruitsFacade.java` and implement the two new methods (use *Fix Imports* as before):

```
public Fruits findByName(String name) {
    Query q = em.createNamedQuery("Fruits.findByName");
    q.setParameter("name", name);
    return (Fruits)q.getSingleResult();
}

public Collection getFruitsList() {
    ArrayList outList = new ArrayList();
    try {
        Iterator i = findAll().iterator();
        while (i.hasNext()) {
            Fruits f = (Fruits) i.next();
            outList.add(f.getName() + ", " + f.getPrice() + " kr, " +
                        f.getQuantity() + " in stock");
        }
    }
    catch (Exception e) {
        outList.add(e.toString());
    }
    return outList;
}
```

5. Study the code in `FruitsFacade.java` and make sure you understand at least the big picture. Build the project before you continue.

## 2.3.4 Creating the Servlet

1. Right-click *MyEJBShop-war* and select *New* | *Servlet*…
2. Type “ShopServlet” as the *Class Name* and “servlets” as the *Package*, and click *Finish*.
3. Right-click inside the `processRequest` method, select *Insert Code*… and then *Call Enterprise Bean*... Expand *MyEJBShop-ejb* and select *FruitsFacade*. Select *Remote* as the *Referenced Interface*.
4. Repeat the above step, but now select *Local* as the *Referenced Interface*.
5. Modify `processRequest` to look something like below. (Implementing the complete shop with shopping cart and GUI is optional, but a lot of fun!)

```
protected void processRequest(HttpServletRequest request,
                              HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet ShopServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet ShopServlet at " +
                    request.getContextPath () + "</h1>");

        out.println("<h2>Contents of the Fruit Shop</h2>");
        Iterator i = fruitsFacade.getFruitsList().iterator();
        while (i.hasNext())
            out.println(i.next() + "<br>");

        out.println("<h2>Test of the Local Interface</h2>");
        out.println("There are " + fruitsFacade1.count() +
                    " different fruits<br>");
        try {
            Fruits f = fruitsFacade1.findByName("Banana");
            out.println("The price of bananas is " +
                        f.getPrice() + " kr");
        }
        catch (Exception e) {
            out.println("Could not find the price of bananas (" +
                        e.toString() + ")");
        }

        out.println("</body>");
        out.println("</html>");
    }
    finally {
        out.close();
    }
}
```

## 2.3.5 Deploying the Application

1. Right-click *MyEJBShop* and select *Properties*. Select the *Run* category, type “ShopServlet” as the *Relative URL*, and click *OK*. This means that the application will start by opening the servlet instead of the default *index.jsp*.
2. Press F6 to deploy and run the application. Note that the application and database servers are started automatically and will be stopped when you exit NetBeans.