

Distributed Systems: Java EE session 1

Wouter De Borger, Stefan Walraven, Bert Lagaisse,
Steven Op de beeck, Fatih Gey, and Wouter Joosen

October 13, 2015

Overview

There will be 3 exercise sessions on Java EE:

- 13/10/15 in PC lab:
 - Introduction to Java EE and session beans.
 - Debugging in the context of Java EE.
 - **Submit** your code on Toledo **before Friday, October 16, 19:00**.
- 03/11/15 in PC lab:
 - Introduction to Java EE persistence.
 - **No code to submit.**
- 10/11/15 in PC lab:
 - Java EE persistence and transactions.
 - **Submit** the final version of your code on Toledo **before Friday, November 13, 19:00**.

In total, **each** student must submit 2 assignments to Toledo.

1 Introduction

Goal: The goal of this session is familiarizing yourself with building, deploying and running a basic Java EE application. You have three tasks in this session: 1) deploying and running an existing Java EE application, 2) extending this existing application, and 3) using the debugger.

Approach: This session must be carried out in groups of *two* people. Team up with the same partner you collaborated with in the previous sessions.

Submitting: On Friday, at the end of the week of the session, **each** student must submit his or her results to the Toledo website. To do so, *clean your project*, and create a zip file of your “CarRental” project using the following command:

```
zip -r firstname.lastname.zip <your project directory>
```

Then submit the zip file to the Toledo website (under Assignments) before Friday, 19:00.

Important:

- When leaving, make sure your server is no longer running.
- NetBeans projects have a **nbproject** directory. In this directory, there is a **private** sub-directory. When moving NetBeans projects to other machines, make sure to remove all **private** folders in all projects.
- Retain a copy of your work for yourself, so you can review it before the exam.

2 Preliminaries

Start the NetBeans IDE by executing the following command:

```
/localhost/packages/ds/netbeans/bin/netbeans
```

Important: Check if you have enough storage space in your home directory (via **quota**).

Create a new domain. A domain is an instance of the application server bound to this machine. When moving to another machine, a new domain must be made. For performance reasons, it is paramount that the domain is stored on the local hard disk and not in your home folder.

In NetBeans, navigate to the **Services** tab and open the **Servers** dropdown. Remove all servers listed. Right-click on **Servers** and choose **Add Server**. A wizard window will open.

1. The new server is a **GlassFish Server 4.1**. Feel free to choose a cool name for your server.
2. Set the server location to

```
/localhost/packages/ds/glassfish
```

and create a **Local Domain**. A domain is an instance of the Java EE server, and each domain has its own configuration, log files, applications etc.

3. Store the domain in a subfolder of **/tmp**. In the box **Domain** enter **/tmp/<your domain dir>** with an appropriate name for your domain folder, and click **Finish**.

A new domain with its own configuration, logs and applications is created. All files relevant to the domain can be found in the **<your domain dir>** subfolder you selected in step 3. The domain is not running yet, but will be started automatically by NetBeans when you deploy an application.

Open the Java EE car rental application. Download **ds_jee_1.zip** from Toledo and extract the contents of the zip file into your home directory. The zip file contains the Java EE car rental application (a NetBeans project). Open this Java EE “CarRental” project (**File** → **Open Project**) and **choose to include the required subprojects**.

3 The Car Rental Application

The goal of the car rental application is to allow clients to remotely rent cars at multiple car rental companies. Currently, only part of the functionality is implemented and it is up to you to complete it in this session (see Section 4). This functionality differs from the Java RMI application in the first exercise session. In the first subsection we will briefly discuss this extended functionality of the car rental application. The next three subsections describe the three components the car rental application consists of: a session bean, an application client, and common code. The last subsection provides instructions to run the car rental application.

3.1 Extended functionality

In the first exercise session (Java RMI), the goal was to develop a distributed reservation system for **one** rental company. In this exercise session, however, the result has to be a distributed booking system for a **car rental agency**. A client now has the opportunity to make several reservations at multiple rental companies via this rental agency. Therefore, every client needs a session to keep all the conversational state, for example the tentative reservations (i.e. quotes). When confirming these quotes, either all or none of them are finalized. The session shields the client from this complexity by executing business tasks inside the server.

3.2 Session

The `CarRentalSession` is a server-side Enterprise Java Beans (EJB) component. More specifically, it is a stateful session bean with a remote interface:

- The component is a *session bean*, since it represents a session with the client. A session bean exists only for the duration of a single session.
- The component is a *stateful* session bean, meaning that it retains state between method invocations. Contrary to stateful session beans, stateless session beans retain no state between method invocations. The `@Stateful` annotation (see class `CarRentalSession`) is used to indicate a component is a stateful session bean.
- The component has a *remote* interface, since remote clients (i.e. clients running in a different VM) need to be able to use the component. The Java EE server uses RMI under the hood to implement this remote communication. One can give EJB components a local interface when the bean is only used by local clients to avoid the overhead introduced by RMI. The `@Remote` annotation (see interface `CarRentalSessionRemote`) is used to indicate an interface represents a remote interface.

The package `session` contains the code for the the session bean itself, while the package `rental` contains the helper classes (regular Java code).

3.3 Application Client

The application client is a client-side component, and uses the session bean's remote interface to make reservations. The main method of the application client uses the static field `session`, but never initializes it. The `@EJB` annotation indicates that the application client container is responsible for initializing this field. More specifically, the application client container will initialize this field by performing a JNDI lookup. This is called *dependency injection*. Note that server-side components too can depend on dependency injection to look up other components or resources.

In situations where injection is not sufficient, JNDI can be directly called to manually obtain a session. The simplest solution is by manually performing the JNDI lookup as internally done when using the `@EJB` annotation without arguments:

```
InitialContext context = new InitialContext();
session = (<BeanInterface>) context.lookup(<BeanInterface>.class.getName());
```

However, this approach as well as using `@EJB` without arguments fail to acquire EJB references when no Java EE container is available (e.g. Java SE client) or when used in a different Java EE application (other ear file). Therefore every container must assign a (*portable*) *global JNDI name* to EJBs. For more information about portable global JNDI names, we refer to http://blogs.oracle.com/MaheshKannan/entry/portable_global_jndi_names.

While the application client in our example is just a command-line program, it is possible to create a full-fledged GUI (not part of the assignment).

3.4 Common Code

All classes that are used on both the server and the client are placed in the project `CarRental-lib`. This allows NetBeans to package all common classes into a JAR file that is deployed to client as well as server. **Make sure to store all shared classes into this project!**

3.5 Build, Deploy and Run

Before we can run our program, we need to go through several steps.

First of all, the Java source files are compiled to class files with a regular Java compiler. To do so, right-click `CarRental` and click **Build**. The build command also packages the class files into jar (and ear) files. In addition to the class files, these jar files contain *deployment descriptors*, i.e. XML files containing configuration information for the component.

Deploying a Java EE application consists of uploading the necessary components to the Java EE server. To do so, right-click `CarRental` and click **Deploy**. Note that the application server is automatically started by NetBeans when deploying the application. The application server provides a web interface to inspect and update its configuration. Simply point your browser to `localhost:4848` (default user name and password are used).

We are now ready to run our application client. Return to NetBeans, right-click `CarRental` and click **Run**. If everything is right, the output window should contain the following output:

```
found rental companies: [Dockx, Hertz]
```

4 Assignment: Extending the Car Rental Application

4.1 Making Reservations

The given car rental application currently only allows clients to list the names of all registered car rental companies. We would like clients to be able to make reservations. To do so, add the methods `createQuote`, `getCurrentQuotes` and `confirmQuotes` to the session bean. The method `createQuote` tries to make a quote (`Quote`) for a given constraint (`ReservationConstraints`) at a particular car rental company. *All quotes must be stored in the session.* The method `getCurrentQuotes` returns all quotes in the current session. The method `confirmQuotes` effectively ties these quotes to a car and updates the reservations corresponding to that car. If one of the quotes cannot be finalized, cancel all reservations and raise an exception (`ReservationException`).

In the current exercise session, the different car rental companies are simply stored in a static field. To reset the value of the static field, simply redeploy the application. In the next session, we will persist the car rental companies and their reservations in a database.

4.2 Manager

Add a new stateless session bean with a remote interface. The session bean should enable managers to request (i) a list of car types for a particular car rental company, (ii) for each car type (in a particular car rental company) the number of corresponding reservations, and (iii) the number of reservations made by a particular client.

Important design consideration: What information is sent to the client and why? What would happen if the information was altered on the client side? Consider your alternatives.




4.3 Application Client

Update the application client (i.e. `Main` class) so that it extends `AbstractTestAgency` and implement the inherited methods¹. Add extra operations when necessary, even if not explicitly described in the assignment. *The abstract class may contain redundant parameters*, depending on your implementation of the server application. Run the test `simpleTrips`.

¹When dependency injection is insufficient to obtain a session bean, you can fall back to JNDI (cf. Subsection 3.3).

5 Debugging

Debugging distributed applications is slightly more complex than debugging single-process applications. Two (Java virtual) machines are involved (i.e. client and server), therefore two different debuggers are used. The current version of NetBeans starts both debuggers automatically.

- By clicking in front of a line of code, you can put a breakpoint on that line during execution. Put a breakpoint at some point of interest in the car rental client code.
- Use CTRL+F5 to start your application in debugging mode with NetBeans (or hit ) . The client application and the server start automatically in debug mode.
- NetBeans will remark to “Use 9009 to attach the debugger to the Glassfish Instance”.
- Choose to attach the debugger via the menu **Debug** → **Attach Debugger**. Set the debugger to **JPDA debugger**. The connector should be **SocketAttach**, because the debugger communicates with the server using socket-based communication. Set the host to “localhost” and the port to “9009”, and click OK. This connects the debugger to the Server VM.
- A debugging tab will show all active threads. On top of this tab is a selection box to switch between the client and Server VM.
- The execution will stop at a breakpoint and you can inspect the execution context of the breakpoint. Variables that have a value in the execution context of the breakpoint can be inspected by hovering the mouse over the variable. You can also inspect the local variables and deeper object structures in the **local variables** tab window. Use F5 (or ) to continue to the next breakpoint. You can also use the navigation buttons to step through the code.
- Hit Shift+F5 (or ) to stop debugging.

6 Practical Information

In case of problems, try the following things:

- Undeploy all applications from the application server in the services tab (especially the client application). Restart the application server.
- Check if you have enough storage space in your home directory (via `quota` and `du | sort -n`).
- Remove all your old NetBeans configuration and settings by deleting the folders `.netbeans`, `.netbeans-derby` and `.cache/netbeans` in your home directory.
- When you get a pop-up to unlock the login keyring, fill in your login password. When this fails, remove everything under `.gnome2/keyrings/`. Next time the pop-up shows up, fill in your login password.

7 Conclusion

After this session, you should be able to answer the following questions:

- What is a session bean? What is the difference between a stateful and a stateless session bean?
- What is the difference between the local and remote interface of an EJB component?
- What is JNDI? How is JNDI used in the context of Java EE? What is dependency injection?
- Why would anyone choose a Java EE solution over a regular Java SE application with Java RMI?

References

[API] Oracle. *JavaTM Platform, Enterprise Edition 7 API Specification*. <http://docs.oracle.com/javase/7/api/>

[Tutorial] Oracle. *Java EE Tutorial*. <http://docs.oracle.com/javase/5/tutorial/doc>

Good luck