

## Java EE: assignment 1

# Leveraging services from component-based middleware for distributed applications

---

## Practical arrangements

There will be 3 exercise sessions on Java EE:

1. Tuesday 3 November 2020:
  - Assignment 1 (*this assignment*): Introduction to Java EE and session beans; debugging in the context of Java EE.
  - **Submit** your code on Toledo **before Friday 6 November 2020, 19:00**.
2. Tuesday 10 November 2020:
  - Assignment 2: Java EE persistence and transactions.
  - No submission.
3. Tuesday 17 November 2020:
  - Assignment 2: Java EE persistence and transactions (continued).
  - Submit the final version of (1) *the report* and (2) *the code* on Toledo before Friday 20 November 2020, 19:00.

In total, **each** student must submit their solutions to the two Java EE assignments to Toledo.

**Submission:** Before Friday 6 November 2020 at 19:00, **each** student must submit their results to the Toledo website. To create this zip file in NetBeans, in the File menu, select Export Project → To ZIP . . . , then use Other Directory to browse to the folder that contains all four (sub)projects. (Root Project is insufficient: it will only include one of four projects!) Name your zip file `jee1.firstname.lastname.zip`.

Alternatively, you can manually create a zip file of your CarRental project. First, clean your project through NetBeans. Then, remove the `nbproject/private` folder from all four subprojects. Finally, execute the following command in the terminal:

```
zip -r jee1.firstname.lastname.zip <your project directory>
```

Once you have created a zip file, submit it to the Toledo website (under Assignments) before the deadline stated in the overview above.

**Important:**

- This session must be carried out in groups of *two* people. Team up with the same partner you collaborated with in the previous sessions.
- Retain a copy of your work for yourself, so you can review it before the exam.
- When leaving, make sure your server is no longer running.
- NetBeans projects have a `nbproject` directory. In this directory, there is a `private` subdirectory. When moving NetBeans projects to other machines, make sure to remove all `private` folders in all projects.

Good luck!

# 1 Introduction

Java Enterprise Edition (Java EE) is a middleware that adds services relevant for distributed (web) systems to the Java Standard Edition (Java SE). It provides services such as session management, persistence and transactions through components that reflect the tiers within an application, allowing for cleanly structuring business logic, client interaction and back-end processing. Java EE is a powerful model for developing enterprise-grade Java applications, and has evolved into frameworks such as Spring<sup>1</sup> and Hibernate<sup>2</sup>.

**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 application using services provided by the middleware, and 3) familiarizing yourself with the debugger.

## 2 The Car Rental Application

We continue the theme of the second Java RMI assignment, where our car rental application supports (remote) reservations at multiple car rental companies. Currently, only part of the functionality is implemented, and it is up to you to complete it in this lab session (see Section 3). The next subsections describe the rental agency, project structure, application client, and car rental sessions.

### 2.1 Rental agency

In contrast to the RMI assignment, where booking data was stored at individual car rental companies, our central rental agency will now fully handle the infrastructure of the car rental companies that it has a contract with. This means that processing bookings and storing the appropriate data is now centralized at the agency. Therefore, clients only need to interact with this central agency to make several reservations at multiple, regional rental companies.

The rental agency needs to store some information about the car rental companies in order to perform its operations. More precisely, in this iteration of the assignment, the agency will keep a mapping with the `CarRentalCompany` objects of the contracted companies. The data related to each car rental company will be stored in these objects, without any persistence to permanent storage. In the next lab session, we will use a separate database to improve this data management.

### 2.2 Project structure

The project for the car rental application is structured into four subprojects:

1. `CarRental` is the NetBeans project that ties all subprojects together. It doesn't correspond to a specific container/tier.
2. `CarRental-ejb` contains the server-side session beans, where the conversational state is stored.
3. `CarRental-client` contains the application client, i.e. the client-side component through which reservations are made.
4. `CarRental-lib` contains all classes that are used on both the server and the client. 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 classes shared between the client and the server — but not everything — in this subproject!**

### 2.3 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

---

<sup>1</sup><https://spring.io/>

<sup>2</sup><https://hibernate.org/>

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<sup>3</sup> lookup. This is called *dependency injection*. Note that server-side components (e.g. beans) also can depend on dependency injection to look up other components or resources.

In situations where dependency injection is not sufficient, for example when multiple simultaneous stateful sessions are used at the same client, JNDI can be directly called to manually obtain a session. The simplest solution is to manually perform the JNDI lookup, as is done internally 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. in an application client implemented in Java SE) or when used in a different Java EE application (i.e. another ear file). Therefore, every container must assign a (*portable*) *global JNDI name* to EJBs, which can then be looked up in the InitialContext to retrieve a reference to the EJB.

For more information on portable global JNDI names, we refer to [https://archive.md/20121217170521/https://blogs.oracle.com/MaheshKannan/entry/portable\\_global\\_jndi\\_names](https://archive.md/20121217170521/https://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).

## 2.4 Reservation sessions

Every car renter needs a session to keep all their conversational state, for example the tentative reservations (i.e. quotes). The session abstracts away managing this complexity at the client by storing state and executing business tasks inside the server.

The ReservationSession 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. The @Stateful annotation (see class ReservationSession) 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, without exposing the implementation (which also allows for updates to the server-side implementation without requiring a client-side update). The Java EE server uses RMI under the hood to implement this remote communication. The @Remote annotation (see interface ReservationSessionRemote) is used to indicate that an interface represents a remote interface.

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

## 3 Assignment

### 3.1 Implementation

#### 3.1.1 Car renters: 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 following methods to the server-side ReservationSession session bean:

---

<sup>3</sup>JNDI is a naming service, comparable to but more generic than the RMI registry.

1. The method `getAvailableCarTypes` allows car renters to check the availability of car types in a certain period, across all car rental companies and regions.
2. The method `createQuote` tries to make a `Quote` that satisfies the given `ReservationConstraints`. If multiple companies can provide a quote within the given constraints, *any one* of them can be offered to the customer. *All quotes must be stored in the session.*
3. The method `getCurrentQuotes` returns all quotes in the current session.
4. The method `confirmQuotes` effectively ties these quotes to a car and updates the reservations corresponding to that car. When confirming these quotes, either all or none of them are finalized. If one of the quotes cannot be finalized, cancel all reservations and raise an exception (`ReservationException`).

You do not need to do any authentication of car renters (or configure car renter users in the application server): anyone using the application client is authorized to create a session and make reservations.

### 3.1.2 Manager

As in the RMI assignment, we want to support functionality for the manager of the car rental agency to request statistics on the different companies contracted with the agency. Therefore, add a new *stateless* session bean with a remote interface for the sessions of this manager. Contrary to stateful session beans, stateless session beans retain no state between method invocations.

The session bean should enable managers to request:

1. the number of reservations for a particular car type in a particular car rental company, and
2. the number of reservations made by a particular client across all companies managed by the agency.

**Important design consideration:** What information is sent to the client and why? Consider your alternatives.

Ensure that you enforce appropriate security measures in your code such that only authorized users (with the `Manager` role) can acquire a manager session and request these statistics. Specify the appropriate permissions using annotations (only) where necessary.

**Note:** When configuring the realm in the GlassFish application server, you can use any combination of username and password for authorized users, as long as you make sure that they will have the relevant role with the name `Manager` (e.g. by assigning it to a `Manager` group and enabling the default mapping of principals to roles). One user is sufficient, and you do not need to share these credentials with us (we add our own user when checking your assignment).

### 3.1.3 Application Client

Update the `Main` class of the application client so that it extends `AbstractTestAgency` and implement the inherited methods<sup>4</sup>: reading the `JavaDoc` available in the `AbstractTestAgency` class helps to understand the operations of the `simpleTrips` script. Add extra operations when necessary, even if not explicitly described in the assignment. The methods of the abstract class may contain redundant parameters, depending on your implementation of the server part of the car rental application. Run the `simpleTrips` script to test your implementation.

## 4 Getting up and running

### 4.1 Setting up the NetBeans IDE and GlassFish Server

It is advised to use the JEE edition of NetBeans 8.2 in this assignment. For more details on the installation of NetBeans and the GlassFish application server, consult the BYOD guide (on Toledo).

**Start the NetBeans IDE.** Execute the following command:

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

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

<sup>4</sup>When dependency injection is insufficient to obtain a session bean, you can fall back to JNDI (cf. Subsection 2.3).

**Set up the GlassFish server.** This application server implements the Java EE APIs, provides the middleware services to the components, and takes on the responsibility of serving application data to clients of your Java EE application.

You set up GlassFish by creating a new domain. A domain is an instance of the application server bound to a particular machine. When moving to another machine, a new domain must be made. As your server will often read from/write to disk, when using the computers in the labs, it is important for performance reasons 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 listed servers. Right-click on Servers and choose Add Server. A wizard will open.

1. The new server is a GlassFish Server 4.1.1. 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. Next to the DAS Port and HTTP Port boxes, make sure to tick Default. Finally, click Finish.

This creates a new domain with its own configuration, logs and applications. 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\_je\_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 the CarRental project (File → Open Project) and **choose to include the required subprojects**.

## 4.2 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. The application server is automatically started by NetBeans when deploying the application. Note that, after deployment, you can therefore already use the web interface provided by the application server to inspect and update its configuration: simply point your browser to localhost:4848 (default username and password are used).

Once you are ready to run your 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]
```

In the current assignment, the different car rental companies are stored in a static field at the agency. To reset the value of the static field, redeploy the application.

If you have implemented methods that verify whether a user is authorized to call the method and no user is authenticated yet, NetBeans will display a popup asking to enter a username and password. Try various credentials to test all possible flows of your method, e.g. valid credentials of a user with the required role for testing the functionality, invalid credentials for testing whether authentication is correctly implemented, or credentials of a user without the required role for testing whether authorization is correctly implemented.

### 4.3 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, there is a selection box to switch between the client and server VMs.
- 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.

### 4.4 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.
- NetBeans projects have a `nbproject` directory. In this directory, there is a private subdirectory. When moving NetBeans projects to other machines, make sure to remove all private folders in all projects.
- 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.
- You may get an error that NetBeans is ‘Unable to delete directory’, in particular when NetBeans crashed or was forcibly closed. This is due to a process having a lock on that directory; often, this lock is held by a GlassFish server instance still running in the background. To kill this server process, in Windows, you need to kill it via Task Manager (search for Java processes); in MacOS and Linux, you can run the following command in your Terminal:

```
kill $(ps aux | grep "glassfish" | grep -v 'grep' | awk '{print $2}')
```

Afterwards, restart NetBeans and the GlassFish server.

## References

- [1] Oracle. *Java™ Platform, Enterprise Edition 7 API Specification*. <https://docs.oracle.com/javaee/7/api/>
- [2] Oracle. *Java EE Tutorial*. <https://docs.oracle.com/javaee/7/tutorial/>