Java EE: assignment 2

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: 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 *(this assignment)*: Java EE persistence and transactions.
   - No submission.
3. Tuesday 17 November 2020:
   - Assignment 2 *(this assignment)*: 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 20 November 2020 at 19:00, *each* student must submit a zip file 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!) Create the zip file and name it `jee2.firstname.lastname.zip`.

Alternatively, you can manually create a zip file of your submission. 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 jee2.firstname.lastname.zip <your project directory>
```

Make sure that the report PDF is also part of your submission: add it to the main directory of your solution for the final zip-creation process to include this report, add it manually to the zip file, or upload it separately on Toledo.

Once your submission is ready, 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.
- When leaving, make sure your server is no longer running.
- NetBeans projects have a `nbproject` directory containing a `private` subdirectory. 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.

Good luck!

1 Introduction

Goal: The goal of this assignment is to design and implement an application that uses additional services provided by the Java EE middleware, namely persistence and transactions, in order to improve how company data is maintained over time and how concurrent bookings are managed.

2 The Car Rental Application

2.1 Rental agency

In the previous assignment, our central rental agency loaded and stored the data of the car rental companies — with which it had a contract — into `CarRentalCompany` objects on startup. However, this requires restarting the agency server to reload this data, and leads to data loss if the server crashes.

We therefore move towards persisting the company data in a database. This database is still managed centrally by the rental agency, and houses all the data for all the companies. We want to ensure that no service interruption occurs when any new car rental companies need to be added, or when existing companies need to be updated. The manager of the rental agency should therefore be able to dynamically create new companies and load and update their data in the database.

3 Assignment

3.1 Implementation

The goal of this assignment is to modify the given application such that the car rental companies, cars and reservations are stored in a database. More specifically, the following functionality should be provided:

1. The classes `CarRentalCompany`, `Car`, `CarType`, and `Reservation` should be changed to `Entity` classes, and the appropriate relationships between the entities should be defined. After doing so, you should be able to delete the class `RentalStore`. However, it's best to keep a copy as you might be able to reuse its data loading code.
2. Add a way for managers to (dynamically) add car rental companies, car types and cars to the database from the given `.csv` files. Extend the client so that, when starting up the agency server, it uses this management interface to load the car rental companies and their cars (and their respective car type) into the database. Ensure that *no service interruption* occurs (i.e. making reservations is temporarily unavailable) when any new car rental companies or cars need to be added afterwards.
3. Car renters should be able to make reservations at multiple car rental companies. A list of `Quotes` (i.e. tentative reservations) should be kept on the server for the duration of the car renter's session. It is not necessary to make this list persistent. Once a car renter decides to confirm their quotes, all tentative reservations should be effectively stored in the database. Ensure that no overlapping reservations for the same car can be created. If one of the reservations fails (e.g. because no free car of the specified car type is available anymore), all reservations corresponding to the session should fail (rollback). Use the transaction support of Java EE. *Avoid using any transactions when the functionality doesn't require it!* In the end, quotes can be deleted once tentative reservations are confirmed and persisted in the database.
4. Managers of the rental agency should be able to query information and retrieve different statistics to support customer profiling and advertising. Because of efficiency reasons, it is recommended in Java EE to use persistence queries (JPQL) without additional application logic in order to retrieve the information of interest. To query a specific entity with a given key, `em.find()` is the recommended approach.
   Therefore, use JPQL as much as possible when retrieving the following information (methods with an implementation that doesn't use persistence are already present in the provided session beans). We **require** you to use JPQL for the following queries:
   (a) look up the names of all car rental companies (`getAllRentalCompanies`)
   (b) look up all car types in a company (`getCarTypes`)

(c) look up the IDs of all cars of a particular type in a company (`getCarIds`)
(d) retrieve the number of reservations for a particular car in a car rental company (`getNumberOfReservations(company, type, id)`)
(e) *idem* for a particular car type (`getNumberOfReservations(company, type)`)
(f) look up the available car types in the given period (`getAvailableCarTypes(start, end)`)

You must also support the following functionality, adding appropriate methods to classes where necessary. We encourage you to implement these queries **as much as feasible** using JPQL: a solution that uses JPQL more extensively (even exclusively) can earn you bonus marks. However, make sure that the functionality to retrieve the following data is supported in one way or another:

(g) retrieve the number of (final) reservations made by a particular car renter
(h) retrieve the best client(s) across car rental companies (i.e. highest total of reservations)
(i) retrieve the most popular car type of a car rental company for a given calendar year (the start date of the reservation counts)
(j) retrieve the cheapest car type available given start date, end date and region

5. Like in the previous assignment, ensure that you enforce appropriate security measures in your code such that only authorized users (with the `Manager` role) can acquire a manager session, load data into the database and request the customer statistics.

Note:

1. Depending on your implementation strategy, methods might have redundant parameters, you may need to add or move classes, methods, files, …, and/or you may need to make other changes. Be sure that you are able to motivate why the changes you make are necessary.
2. Throughout this assignment, always consider **good software engineering and coding practices**. More specifically: think about *where* to implement new behavior, respect *encapsulation*, use the *proper arguments*, *don't scatter JPQL queries* throughout the code, etc.

3.2 Reporting

We would like you to report on your design and the decisions you have made, and on your understanding and implementation of distributed concepts. Provide your answers in English, and keep them short, concise and to the point: a few lines is often enough. You are encouraged to use the LaTeX template on Toledo. Don't forget to put your names in the report, and store the report (as one PDF called `report-jee.pdf`). Make sure that this PDF is part of your submission: add it to the main directory of your solution for the final zip-creation process to include this report, add it manually to the zip file, or upload it separately on Toledo.

Compile a report that answers the following questions:

1. Outline the different **tiers** of your application, and indicate where classes are located.
2. Why are client and manager session beans **stateful and stateless** respectively?
3. How does **dependency injection compare to the RMI registry** of the RMI assignment?
4. **JPQL persistence queries** without application logic are the recommended approach for retrieving rental statistics. Can you explain why this is more **efficient**?
5. How does your solution compare with the Java RMI assignment in terms of **resilience against server crashes**?
6. How does the Java EE middleware reduce the effort of **migrating to another database engine**?
7. How does your solution to concurrency prevent **race conditions**?
8. How do **transactions compare to synchronization** in Java RMI in terms of the **scalability** of your application?
9. How do you ensure that **only users** that have specifically been assigned a **manager role** can open a ManagerSession and access the manager functionality?
10. Why would someone choose a **Java EE** solution over a regular **Java SE** application with Java RMI?

4 Getting up and running

4.1 Starting NetBeans

1. Start NetBeans by executing the following command:

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

2. Set up a domain, as described in the first assignment.
3. Download `ds_jee_2.zip` from Toledo. Extract the contents of the zip file. The zip file contains the Java EE car rental application (a NetBeans project). You may start from the provided code to develop your project.
4. Open the `CarRental` project, as described in the first assignment.

4.2 Important Remarks

- You have to create a new persistence unit before you can run an application using persistence entities. To create a new persistence unit, select the `CarRental-ejb` project, right-click the project name and choose `New` → `Other...`. In the "New File" window, choose `Persistence` → `Persistence Unit` and click "Next". Select `jdbc/sample` as your data source, keep the default persistence provider, and choose **"drop and create"** as your table generation strategy. Make sure the persistence unit uses the Java Transaction API. Click "Finish". A `persistence.xml` file is created in the folder "Configuration Files". Also make sure to explicitly include all entities in the list **"Include Entity Classes"**. Otherwise the application server may be incapable of detecting the entities in the `CarRental-lib` project.
- When running the application client within NetBeans, it will redeploy the application and reinitialize the database each time you click `Run`. However, do not continuously try to deploy your application when your (server) implementation is not finished. *"Trial-and-error" is bound to go wrong!*
- When defining relationships, do not forget to set the `CascadeType` and `FetchType` when necessary. Check the default settings.
- Fields of type `java.util.Date` in entity classes must be annotated with the `@Temporal` annotation. The correct `TemporalType` is `DATE`.
- Often it is useful to have Java EE generate primary keys for you. To do so, annotate the primary key field with `@GeneratedValue` with `AUTO` or `IDENTITY` as the strategy.
- Do not fully rely on NetBeans (e.g. warnings in yellow): you have to be able to explain everything!
- To inspect the database structure, open the Services tab and under "Databases", right-click on `jdbc:derby://localhost:1527/sample [app on APP]` and select 'Connect'. In the `APP` database, you will find the tables for your project.

Important: 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 containing a `private` subdirectory. When moving NetBeans projects to other machines, make sure to remove all `private` folders in all projects.
- If you get a pop-up to "unlock the login keyring", enter your login password. When this fails, remove everything under `.gnome2/keyrings/`. Next time you get the pop-up, enter your login password.

5 Documentation

Several sources of documentation are useful, apart from the slides and the textbook:

- Java EE 5 tutorial: `https://docs.oracle.com/javaee/5/tutorial/doc/` (most extensive)

- Java EE 7 API documentation: `https://docs.oracle.com/javaee/7/api/`
- Java EE example applications are available at `/localhost/packages/ds/javaeetutorial5/`. The most instructive example is the `Roster` application. To use the project, you need to copy the master project (shared by all examples) and the `Roster` project to a writable location (for example your home directory), e.g. with the following command: `/localhost/packages/ds/javaeetutorial5/copyExample.sh ~ ejb/roster`. Then the project can be opened in NetBeans.