Lab session 8

Targets

- Rebuild the CWS application using the domain model pattern and a JPA mapper.
- Evolve the layered architecture from the previous JDBC version to the hexagonal architecture with a domain model and the mapper.

Tasks

Evolve the architecture from JDBC version

Please, follow this workflow:

- Import the Eclipse project for this session.
- Copy your JPA mapped domain classes from the previous session into the package `uo.ri.cws.domain`.
- Start up the attached HSQLDB database.
- Observe the project distribution. There is one project per actor (fully functional) plus the services project.
- See the service layer interfaces, they are the same as for the previous delivery. Observe the use of the patterns service, façade, factory and DTO.
- Locate the equivalent *Transaction Script* classes. They are empty. We have to redo all the functionality (package `uo.ri.cws.application.service.*.*`). Now they must be done using the domain model classes and the repositories. As a consequence, they are much reduced.

We will start with the CRUD operations for mechanic management (manager actor). After that, solve the Invoicing use case for the cashier actor.

Centralize the transaction management

- Add the interfaces *`Command`* and *`CommandExecutor`*.
- Make all the *Transaction Script* implement the interface *`Command`*. Remove all the transaction control code from them.
- Modify the façade implementing classes so that the `CommandExecutor` now executes all the commands. One line of code for each method.
- Locate the utility class *`Jpa.java`* on the *infrastructure.persistence* package. Have a look at it. Please, ensure you understand the purpose of the public methods.
- Implements the `CommandExecutor`. Use the utility class *Jpa.java*.
- Locate the `uo.ri.conf.Factories` class. It acts as a very simple configuration register. Add the `CommandExecutor` factory to it.

- Change every command removing all the redundant transaction control code. They must use the repositories. And the repositories must use the utility class `Jpa.java` to recover the mapper and join to the transaction. Now the commands are very cohesive.

Complete the repositories

- Observe the package `uo.ri.cws.application.repository`, it contains all the repository interfaces needed from the commands. One for each entity. Mind that all inherit from the base interface *Repository*. The repositories follow the metaphor of a collection (add and remove, but not update), they are neither TDG nor DAO.
- The package `uo.ri.cws.infrastructure.persistence.jpa` contains the implementations of those interfaces. Mind there is a base implementation for the common methods. You only must complete the specific queries for each repository. Remember to get the mapper using the *Jpa.java* utility class.
- Add all the queries on the repository implementation classes.
    * Each method executes a query expressed in JPQL that must be externalized on the file `orm.xml`.
    * At least solve the queries of *MechanicJpaRepository*, *InvoiceJpaRepository* and *WorkOrderJpaRepository*.