Enterprise Java Beans

Rui Couto António Nestor Ribeiro

April 27, 2020

Contents

1 Introduction **1**

2 GMS Facade **1**

3 Creating a new EJB **2**
3.1 Adding methods to the bean 3
3.2 Managing persistent sessions 4
3.3 Invoking an EJB method 4
3.4 Next steps 5

1 Introduction

In the previous session a Java Web Application was built. The objective was to develop a web layer, on top of the business and data layers (implemented in previous sessions). Until this point the business layer was developed using normal Java classes (in this sense POJOs, that is, Plain Old Java Objects). The objective of this session is the improvement of the developed solution by adding Enterprise Java Beans (EJB).

2 GMS Facade

The objective of this tutorial is to migrate the façade methods to a EJB, as follows:

1. Register a user;
2. Register a game;
3. Register a platform;
4. List user games;
5. List all games;
6. Search a game;

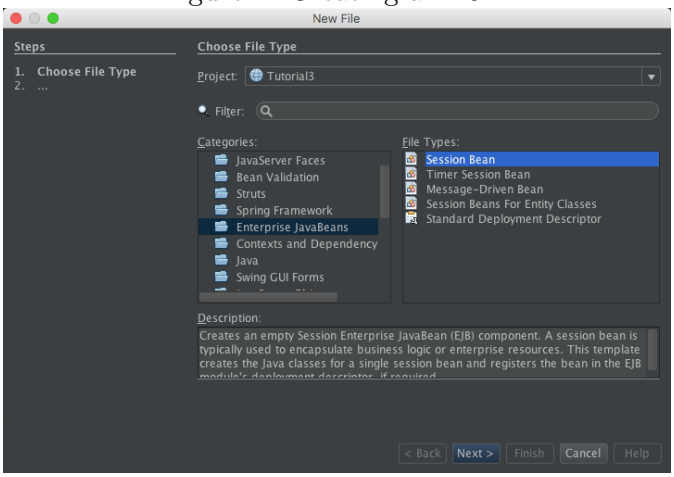
7. Delete a game.

The next section address only the game related methods. As with previous tutorials, provided instructions are valid for the NetBeans IDE and should be adapted to the students current IDE and to the current Java application server.

3 Creating a new EJB

The first step consists in creating the EJB. This process is automated by the IDE. It is suggested to start by creating a new package (e.g. `beans`). Next, right-click in the project, and select `New > Other...`. Under `Enterprise JavaBeans`, select `Session Bean` (c.f. Figure 1).

Figure 1: Creating an EJB.



Since the EJB to provide game related features does not need to hold any state, in the next screen we select `Stateless`. We select also `Create Interface` to create an interface which declares the EJB methods.

After selecting `Finish`, the EJB is created and declared in the application. Note that in the project structure the recently created bean is presented in a specific category. This allows you to easily track the application EJBs.

Tasks

1. Create the GameBean to handle game related methods.
2. Run the application in order to check if everything is working properly.
3. Check that the EJB was deployed[1].

[1] In NetBeans, check in `Services > Servers > Wildfly > Applications > EJB Modules`

Figure 2: Stateless bean and interface.

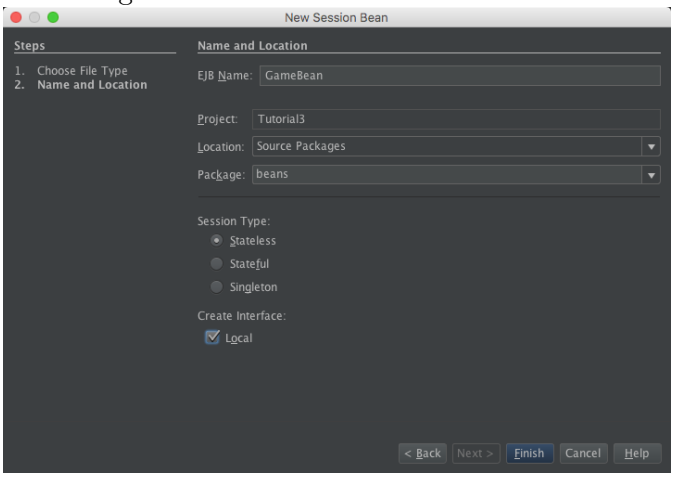
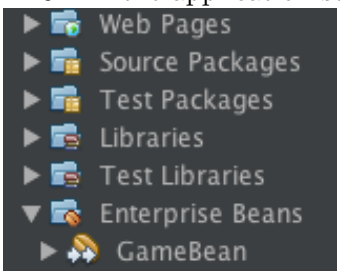


Figure 3: EJB in the application structure.



3.1 Adding methods to the bean

After creating the EJB, and ensuring that everything is working as expected, it is possible to add new methods. In the EJB interface declare the methods to provide. Next follows an "Hello World" example:

```
1 @Local
2 public interface GameBeanLocal {
3     String sayHello(String name);
4 }
```

The method must be implemented in the EJB implementation:

```
1 @Stateless
2 public class GameBean implements GameBeanLocal {
3     @Override
4     public String sayHello(String name) {
5         return "Hello, " + name;
6     }
7 }
```

This method is now available in the EJB, so can be called by an application.

Tasks

1. Add to the EJB the Game related methods (i.e. add a game, list all games, list all games for a user, search a game, delete a game)[2].
2. Validate (e.g. in the Wildfly application server) that the methods are available.

3.2 Managing persistent sessions

The persistency session problem was already addressed in the previous tutorial. Briefly, the databases have a limited number of connections that can be made. Every time an Hibernate session is created a new connection is opened and only closed when that session is destroyed. Sessions left open will consume resources and eventually exhaust the connection pool. So, in order to avoid this problem, the sessions should be preserved.

In the previous tutorial, the persistency session was kept in the servlet. In this case, the session can be managed in the EJB. So, a possible approach is to implement the singleton method. We start by creating a static variable for the session:

```
1 private static PersistentSession session = null;
```

Next, we create a method to get a session instance:

```
1  private static PersistentSession getSession() {
2    if (session == null) {
3      try {
4        session = Tutorial4PersistentManager.instance().getSession();
5      } catch (Exception e) {
6        e.printStackTrace();
7      }
8    }
9    return session;
10 }
```

Note that it is not possible to directly rely upon the static variable, since the EJB is stateless.

Tasks

1. Implement the singleton method as described above.
2. Modify the EJB methods to use the `getSession()` method.

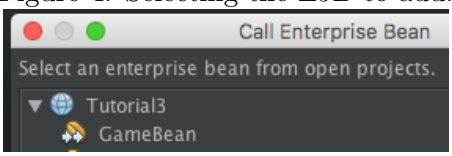
3.3 Invoking an EJB method

At this stage, the EJB methods are ready to be invoked. Netbeans greatly eases the process of invoking them. In any method, right click in the editor, and select `Insert code...`. Next, select `Call Enterprise Bean...`. A new window should

[2]Remember that the EJB is stateless, so it is not possible to rely on memory variables

appear, where it is possible to select the project, and the corresponding EJB (c.f. Figure 4).

Figure 4: Selecting the EJB to add.



After selecting `OK`, the code to invoke the bean is automatically generated. Note the created variable:

```
GameBeanLocal gameBean = lookupGameBeanLocal();
```

as well as the method to perform the lookup.

```
private GameBeanLocal lookupGameBeanLocal() {
  try {
    Context c = new InitialContext();
    return (GameBeanLocal) c.lookup("java:global/Tutorial4/GameBean!beans.GameBeanLocal");
  } catch (NamingException ne) {
    Logger.getLogger(getClass().getName()).log(Level.SEVERE, "exception caught", ne);
    throw new RuntimeException(ne);
  }
}
```

Now, in the code is possible to directly invoke the method:

```
public String localHello(String name) {
  return gameBean.sayHello(name);
}
```

Tasks

1. Follow the presented step to enable the invocation of the EJB methods from the previously implemented `GMS` facade.
2. Modify the existing methods to use the bean methods instead of directly using Hibernate.

3.4 Next steps

Tasks

1. Create the `UserBean` to handle the user related methods.
2. Decide on a strategy to handle `Platform` related methods and implement-it.