The Mandatory Programming Assignment

INF5040 Autumn 2007

The mandatory assignment is to develop a distributed application that models a competition on a rife-range. The system consists of a client, the server that consists of one or more EJB 3.0 beans, a database, and a servlet for presentation of the results.

The client generates, and sends to the server, ten (10) rifle-clubs that participate in the competition, as well as a serie of shots for each team. The serie for each team will consist of ten (10) shots. The shots are randomly generated, with a possible score in the range from 0 to 10 for each shot.

The server stores the data in a database, and later, upon request from the servlet, retrieves them for presentation. We will use the Hypersonic SQL database emedded in JBoss.

Your task is to develop:

- The client
- The servlet for the presentation of the results (you might need one servlet for each of the two alternative ways of handling persistence)
- The server-side EJB 3.0 beans to handle the data about the competing rifle-clubs, each individual shot of each team, and the total score for each team. The data must be persisted in the database.

  Two different solutions with respect to persistence are to be developed:
  1. Use Java Persistence (hint: Entity Manager and entity beans)
  2. Use Java Database Connectivity (JDBC) to access the database (the session bean itself manages persistence).
- Compare the two different ways of handling persistence with respect to performance (time consume) by using an Interceptor for the time measurements.
- In this assignment we will use annotations, there will be no need for XML deployment descriptors (except for the mandatory persistence.xml file).

Remote interface

Use the interface called ShootingResRemote presented below. ***Do not change this interface!***

```
public interface ShootingResRemote
{
  void removeAllData(); // Removes data from a previous competition
  void registerName(int trackNr, String name); // Register a team on a track number
  void registerShot(int trackNr, int shot, int score); // Register serial number and score
                                                       // for a shot on a track
  String getName(int trackNr); // Get the name of a team on a track number
  String getShot(int trackNr, int shot); // Get the result of a shot given the serial and track
                                         //number
  String getTotalScore(int trackNr); // Get the total score of the team on a given track number
  String[] getSortedResultList(); // Returns a sorted result list
}
```

Java persistence-solution

Let a session bean called ShootingResJPBean implement the given remote interface, let one entity bean called Team represent one rifle-club, and let another entity bean called Score represent a shot in the team's series of 10 shots.

Suggested data field for session bean ShootingResJPBean:

```
  @PersistenceContext(unitName="shootingres") private EntityManager manager;
```

Suggested data fields for entity bean Team:

```
  private int id; // The track number
  private String name; // The name of the team
  private Collection<Score> score; // The series of scores for this team
  private int totalScore; // The total score of this team
```

Suggested data fields for entity bean Score:

```
  private int id; // The serial number of the shot
  private int result; // The score of the shot
  private Team team;  // the team that fired the shot
```

JDBC-solution

Let a session bean called ShootingResJDBCBean implement the given remote interface and establish a JDBC connection towards the database.

Let ShootingResJDBCBean have the following data members:

```
  protected Connection c;
  protected Statement s;
  protected DataSource datasource;
```

The session bean must use a number of SQL statements. Below are some examples of SQL statements that may be useful (many of them are incomplete):

- CREATE TABLE CONTESTANT (NO NUMERIC(18,0) NOT NULL PRIMARY KEY, NAME varchar, TOTAL_SCORE numeric)
- CREATE TABLE RESULT (NO numeric, SHOT numeric, SCORE numeric)
- DROP TABLE CONTESTANT IF EXISTS
- DROP TABLE RESULT IF EXISTS
- INSERT INTO CONTESTANT (NO,NAME) VALUES (…)
- INSERT INTO RESULT (NO,SHOT,SCORE) VALUES (…)
- SELECT NAME FROM CONTESTANT WHERE NO = '…'
- SELECT SCORE FROM RESULT WHERE NO = '…' AND SHOT = '…'
- SELECT SUM(SCORE) as TotalScore FROM RESULT WHERE NO = '…'
- UPDATE CONTESTANT SET TOTAL_SCORE = '…' WHERE NO = '…'
- SELECT * FROM CONTESTANT ORDER BY TOTAL_SCORE

Feel free to do changes as they are mere suggestions!

Hints: javax.sql and java.sql may be useful..

Interceptor

Define an interceptor class named Profiler that includes a method

```
@AroundInvoke public Object profile(InvocationContext invocation) throws Exception
```

that measures the execution time of the methods (of the Java Persistence and JDBC solutions) that handle persistence.

Some hints may be found on http://docs.jboss.org/ejb3/app-server/tutorial/interceptor/interceptor.html

Deadline - 16:00 on Friday the 19th of Octber (19.10.07)

(A day later than first said, since the JBoss environments were first up a day later than promissed)

Hand in the following to joakimfi@ifi.uio.no :

- A description of your design and implementation (Word, plain text, or preferrably PDF), which should should include a user guide for compiling and running your distributed application, as well as a discussion of the reasons for possible differences in performance between the two alternative ways of handling persistence
- Everything that is needed to compile and run your distributed application (in a zip or tar.gz archive), such as:
  - source code (should be structured and fairly well commented)
  - build.xml file (preferably only one file for the entire system, that can compile, deploy, execute the client(s), and clean up everyting when finished)
  - persistence.xml
  - client config files (jndi.properties)
  - *(jar-file(s))*
  - *(servlet(s))*
- **Remember:** Include your names, email addresses and group number

Announcements requested by the Department of Informatics

- This assignment is mandatory and must be approved to take the INF5040 exam. The students should work in groups of two or preferrably three.

- All students must read the departmental guidelines for written assignments:
  Norwegian: http://www.ifi.uio.no/studinf/skjemaer/erklaring.pdf
  English: http://www.ifi.uio.no/studinf/skjemaer/declaration.doc