Assignment 5 - OpenEJB

Implement an EJB application consisting of remote session beans, persistent entity beans and a client.

The assignment uses a simple algorithm to measure distance between two nodes in a graph of objects, to mimic working with entities with relationships. Understanding the assignment requires no special knowledge.

Prerequisites

As a technology for client/server architecture and data persistence, the assignment uses the Enterprise Java Beans standard, version 3.0. OpenEJB version 8.0 was chosen as a concrete implementation of the standard. The following knowledge is needed for the implementation:

- Defining remote business interfaces (annotating the interface with the `javax.ejb.Remote` annotation).
- Implementing remotely accessible Stateless Session Bean (annotating the class with `javax.ejb.Stateless`, implementing remote business interface).
- Implementing Entity Beans with persistent attributes (annotating the class with `javax.persistence.Entity`) and specifying the mapping to a relational database via `persistence.xml`.
- Managing persistent Entity Beans via EntityManager injected using the `@PersistenceContext` annotation into the Stateless Session Bean.
- Defining relations between Entity Bean classes (annotations `@OneToMany`, `@ManyToMany` etc.).
- Launching an OpenEJB server, deployment of EJB Beans. Connecting client and server via Naming Service and RMI.

Assignment details

https://d3s.mff.cuni.cz/files/teaching/nswi080/labs/sources-5.zip

The assignment consists of a simple algorithm to measure distance between two nodes in a graph. Graph nodes are represented by objects of the class `Node`.

```java
public class Node
{
    private int id;
    public Collection<Node> getNeighbors();
    public void addNeighbor(Node neighbor);
}
```

The `id` attribute is an unique number identifying the node. The `addNeighbor()` method adds a node to the set of neighbors of the node and is used when a graph is being created. The `getNeighbors()` method returns a set of all neighbors of the node and is used when measuring the distance. The measuring itself is provided by the `Searcher` interface. The interface also provides methods for creating and connecting nodes, so that the client does not work directly with the `Node` objects. The `addNode()` method thus returns the id of the newly created node, the `getDistance()` and `connectNodes()` methods expect the id’s of nodes to work with.

```java
public interface Searcher
{
```
public static final int DISTANCE_INFINITE = -1;
public int getDistance(int nodeFrom, int nodeTo);
public int addNode();
public void connectNodes(int nodeFrom, int nodeTo);
}

Your tasks are:

1. Examine the provided local implementation of the assignment.

2. At first, implement the Searcher as a Stateless Remote Bean and the Node as an Entity Bean with container-managed persistence, and the id attribute as a primary key. Represent the set of neighbors as a suitable relationship between entities.

Modify the client to create nodes and measure distances using a Searcher deployed on the EJB server.

3. Verify that the Node entities are persistent.

   Add a command-line option to the client that will cause it to just run searches on the persisted graph without creating new nodes.

4. Extend the Searcher interface and change the implementation to offer working with persistent nodes to multiple clients simultaneously, so that each client has his own graph.

   The id of the client will be provided as a command line argument.

Consider what is the best way of keeping track of client id (clients should not send their id on each request to the searcher) and how to change the attributes of the Node objects to allow storing independent graphs of multiple clients.

Notes

- EJB server must be running (run-server.sh)

  - Different port when already used
    - Server: ~/OpenEJB/conf/ejbd.properties
      - port = XYZ(XYZ>1024)
    - Similarly admin.properties (for stop-server)
    - Client: props.put(Context.PROVIDER_URL, "ejbd://127.0.0.1:XYZ");

- Server part deployed with run-deploy

  - Needed after each re-compilation!
  - Persistent data stored in ~/OpenEJB/data
    - In case of rather bigger changes it is safer to stop-server and delete the (hsqldb) data

Local implementation

- Class Node
- Interface Searcher and class SearcherImpl
- Launchable class Main (java Main)
• Measure the speed on the random graph

**Searcher as a stateless session bean**

• Searcher as a remote business interface
  ◦ Use appropriate annotation
• Annotation of class SearcherImpl
• Compilation and deployment
  ◦ See scripts in Example
  ◦ Output also contains JNDI name of the bean - Jndi(name=<ClassName>Remote)
• Client - class Main
  ◦ JNDI context creation – see ExampleClient
  ◦ Searcher instance retrieved by JNDI lookup

**Node as an entity bean**

• See Movie and Director in Example
• Annotation of class Node
• Getter/setter for id with appropriate annotation
• Neighbour nodes as relationships among entities
  ◦ Getter/setter with appropriate annotation of the relationship

**Persistence of Node objects**

• Update the class SearcherImpl
  ◦ See ExampleEntityBeans
  ◦ Replace hashmap nodeMap with EJB equivalents
• Annotated EntityManager
  ◦ unitName - corresponds to persistence.xml
  ◦ Method persist() for persistence of created Node
  ◦ Method find() for finding Node by id
• The deployed JAR must contain file META-INF/persistence.xml - see Example
  ◦ Set persistence-unit name and class correctly

**Verify persistence**

• Stop the server after creating the graph, start it before searching through it
• Where to get the node id for the second launch?
  ◦ Try not to assume anything about automatic id assignment to Node
  ◦ Add a method to Searcher that selects a random id from existing nodes

**Multiple graphs**

• Clients with different client id operate on separate graphs
• Change the definition of SearcherImpl to keep track of the client id
  ◦ Do not pass the client id as an argument to every method

**Implementation**

• Reuse available code
• Algorithm implementation of the local variant
• Scripts and code from the example
• Do not add packages etc.
Submission

- By e-mail (deadline is on the web)
- Part of the solution is also documentation of the chosen approach
  - See point 4 in the task description, where you can choose among different approaches
- The submission shall be easy to start
- Make sure it works in the lab
- Do not send any generated files (but send the build script)
- Report issues, ask questions when unclear
  - Mailing list...