The Bank example using Jeremie

April 12, 2002

This document describes step by step how the bank application has been developped with Jeremie, and using the registry through the JNDI interface.

1 The remote interfaces

Here again, we describe the interfaces to be accessed remotely. The first interface is Account[1]:

```
25
26 import java.rmi.Remote;
27 import java.rmi.RemoteException;
28
29 public interface Account extends Remote {
30    double balance() throws RemoteException;
31    String owner() throws RemoteException;
32    void add(double amount) throws RemoteException;
33    void remove(double amount) throws RemoteException, NoCashException;
34 }
```

The second interface is Position[2]:

```
25
26 package bank;
27
28 import Account;
29 import java.rmi.Remote;
30 import java.rmi.RemoteException;
31
32 public interface Position extends Remote {
33    void open(String name, double init) throws RemoteException, OpenException;
34    double close(Account account) throws RemoteException;
35    Account get(String name) throws RemoteException;
36 }
```

[1] contained in `examples/jeremie/jndiBank/srv/Account.java`
[2] contained in `examples/jeremie/jndiBank/srv/bank/Position.java`

2 The server side

The file Server.java[3] contains an implementation of the interface `Account` and `Position` and a `main` method to run the server. The only difference with a similar example written using RMI is the `UnicastRemoteObject` implementation used.

```
25
26 import bank.Position;
27 import bank.OpenException;
28 import java.util.Hashtable;
29 import java.rmi.RemoteException;
30 import org.objectweb.jeremie.libs.binding.moa.UnicastRemoteObject;
```

These are the imports required by JNDI.

```
33 import javax.naming.Context;
34 import javax.naming.InitialContext;
```

AccountImpl is an implementation of Account:

```
37 class AccountImpl extends UnicastRemoteObject implements Account {
38
39    double balance;
40    String owner;
41
42    AccountImpl(String owner, double balance) throws RemoteException {
43       this.balance = balance;
44       this.owner = owner;
45    }
46
47    public synchronized double balance() {
48       return balance;
49    }
50
51    public synchronized String owner() {
52       return owner;
53    }
54
55    public synchronized void add(double amount) {
56       balance += amount;
57    }
58
59    public synchronized void remove(double amount)
60       throws NoCashException {
61          if (amount > balance) {
62             throw new NoCashException("Not enough money in account." +
63                                       " Operation refused.");
64          } else {
65             balance -= amount;
66          }
67    }
68 }
```

[3] contained in `examples/jeremie/jndiBank/srv/Server.java`

PositionImpl is an implementation of Position:

```
71 class PositionImpl extends UnicastRemoteObject implements Position {
72    Hashtable accounts;
73
74    PositionImpl() throws RemoteException {
75       accounts = new Hashtable();
76    }
77
78    public synchronized void open(String name, double init)
79       throws OpenException {
80          AccountImpl account = (AccountImpl) accounts.get(name);
81          if (account != null) {
82             throw new OpenException("Account in name of " + name +
83                                     " already exists. Operation refused.");
84          }
85          if (init <= 0) {
86             throw new OpenException("Account must be opened with a positive" +
87                                     " sum of money. Operation refused.");
88          }
89          try {
90             accounts.put(name,new AccountImpl(name,init));
91          } catch (RemoteException e) {
92             throw new OpenException("Technical problems in opening account." +
93                                     " Operation abandoned.");
94          }
95    }
96
97    public synchronized Account get(String name) {
98       return (Account) accounts.get(name);
99    }
100
101   public synchronized double close(Account account) throws RemoteException {
102         accounts.remove(account.owner());
103         return account.balance();
104   }
105 }
```

The Server class simply contains a main method.

```
108 public class Server {
109
110   public static void main (String[] args){
111      try {
112         Position position = new PositionImpl();
```

The next two lines invoke JNDI and register `position` in a new context. The exact nature of the naming service used doesn't appear in the code.

```
117         Context ctx = new InitialContext();
118     ctx.rebind("Jonathan Position", position);
119         System.out.println("Bank Server ready");
120      } catch (Exception e) {
121         System.err.println("Bank Server exception: ");
```

```
122          e.printStackTrace();
123       }
124    }
125 }
```

3 The client side

The file `Client.java`[4] contains a client for our server. The code is strictly identical to that of a client written for Java RMI.

```
25
26 import bank.Position;
27 import bank.OpenException;
28 import java.rmi.RMISecurityManager;
```

Here again, we have the two JNDI imports.

```
31 import javax.naming.Context;
32 import javax.naming.InitialContext;
35 public class Client {
36    public static void main(String args[]) {
37       try {
38          System.setSecurityManager(new RMISecurityManager());
```

Like in the Server case, an initial context is retrieved. The Position reference is retrieved in that context.

```
42          Context ctx = new InitialContext();
43          Position position =
44             (Position) ctx.lookup("Jonathan Position");
```

The rest of the code is distribution independent.

```
47          System.out.println("Opening Johnny Doe's account with $123.45");
48          try {
49             position.open("Johnny Doe", 123.45);
50          } catch (OpenException e) {
51             System.out.println(e.getMessage());
52          }
53          Account account = position.get("Johnny Doe");
54          System.out.println("Crediting $345.35 to Johnny's account");
55          account.add(345.35);
56          System.out.println("Debiting $635.23 from Johnny's account");
57          try {
58             account.remove(635.23);
59          } catch (NoCashException e) {
60             System.out.println(e.getMessage());
61          }
62          System.out.println
63             ("The balance in Johnny's account is $" + account.balance());
64       } catch (Exception e) {
```

[4] contained in `examples/jeremie/jndiBank/clt/Client.java`

```
65          System.err.println("Bank Client exception: ");
66          e.printStackTrace();
67       }
68    }
69 }
```

4 Compile and run your client and server

- In the `srv` directory:
  - `make` compiles the code and the stubs;
  - `make jrmiregistry` starts the name server;
  - `make server` starts the Bank server;
- In the `clt` directory:
  - `make` compiles the code;
  - `make client` starts the client.