Middleware Labs: JMS

Petr Tůma Vojtěch Horký Antonín Steinhauser
Vladimír Matěna

April 03, 2019

General information

- Task description is on the web

JMS implementation: ActiveMQ

- Download the latest version from Apache (older version is available on the website of the course)
- Unpack into `~/ActiveMQ`

Example

- ActiveMQ infrastructure execution
  - Run in background
    - `$ bash activemq start`
    - `$ bash activemq stop`
  - Or run in foreground
    - `$ bash activemq console`
- Producer/consumer example
  - `ExampleConsumer` consumes messages from two queues
  - `ExampleProducer` sends 'ping' message to those queues
- Running the example
  - `make` script for compilation
  - `consumer` and `producer` scripts for launching the applications

Department of Distributed and Dependable Systems D3S

Task: a simple trading system

- Implement a client of a trading system
- Clients must communicate with each other and with the bank we provide
- Clients publish their lists of goods and buy goods from each other on user request
- The payments go through the bank

Implementation

- Reuse files from the example and `Input-Files/`
- Report issues, ask questions when unclear
- Read the slides
  - To avoid or solve typical problems ;-)
  - Return to them after you have read the instructions
- Code in any editor, run from command-line

Submission

- By e-mail (deadline is on the web)
- Documentation
  - Design and reasoning about the communication protocol used
- 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)

Running

- Use Java **1.8**
  - `java -version`
  - `java-config -s oracle-jdk-bin-1.8`
- ActiveMQ (the broker) must be running prior to the execution of the clients

Implementation notes

- You need two different `Session` instances
  - The first for asynchronous message handling, the second for synchronous (user-triggered) messages and waiting for their replies
  - Single session cannot be used for both synchronous and asynchronous waiting
  - `MessageProducer` from one `Session` should not be used in a different session
  - We need dedicated `MessageProducer` for each `Session`
- Do not forget synchronization of accesses to shared data

Common problems

- Use `equals()` instead of == for `String` comparison

Exceptions at client start-up

- Probably a message in broker queue
- Solution
  1. Stop the bank, client and the broker
  2. Remove directory `data` and `activemq-data`
  3. Restart the broker, bank

Provided parts of the solution

- `Bank.java`: bank implementation
  - Complete, nothing needs to be added (can be studied)
- `Client.java`: skeleton of the client
  - Many parts already prepared
    - JMS initialization, data structures, interaction with the user, the whole communication with the Bank
  - All that is left to do is the communication between clients
    - Sending and receiving goods offers
    - Buying goods (on user's request)
    - Selling goods (asynchronous reaction on other clients' requests)
    - The place marked as `TODO` in the code

Goods offers

- Initialize a suitable channel for transferring offers and create a receiver of its messages
  - Step 1 in the `connect()` method
- Implement sending of offers
  - The `publishGoodsList()` method
- Implement receiving of offers
  - The `processOffer()` method

Buying goods

- Initialize suitable channel for receiving sale requests and create a receiver of its messages
  - Step 2 in the `connect()` method
- Choose suitable message types for communication between clients
  - `MapMessage`? `ObjectMessage`?
- Sending messages requesting a sale
  - Step 1 in the `buy()` method
- Receiving messages requesting a sale
  - Step 1 in the `processSale()` method
- Reserve the requested item
  - Step 2 in the `processSale()` method

Department of Distributed and Dependable Systems
D3S

Buying goods (cont.)

- Accept or refuse the sale
  - Step 3 in the `processSale()` method
- Receive the reply of the sale request message
  - Step 2 in the `buy()` method
- Money transfer request for the Bank
  - Step 3 in the `buy()` method (already implemented)
- After receiving the transaction notification from the Bank (implemented), send a finished sale confirmation
  - Step 3 in the `processBankReport()` method
- Receive the confirmation, notify the user
  - Step 4 in the `buy()` method