Assignment 3 - JMS

Implement an online trading application. Each running instance communicates with other running instances and also with the bank.

The assignment describes a message-based communication protocol that implements a simple trading system. Understanding the assignment requires no special knowledge.

Parts of the implementation are provided. Download source code from https://d3s.mff.cuni.cz/files/teaching/nswi080/labs/sources-3.zip

Prerequisites

The chosen messaging standard is JMS (Java Message Service), using the ActiveMQ 5.15 or ActiveMQ Artemis 2.11 implementation. The following knowledge is needed for the implementation:

- Basic classes and methods provided by the JMS specification, and their application (creating objects of type Connection, Session, Queue, Topic etc.).
- The method of implementing message producers and consumers (MessageProducer and MessageConsumer objects).
- The method of creating and parsing messages of different types (TextMessage, ObjectMessage, MapMessage etc.), knowledge of criteria for choosing an appropriate message type.
- Executing the JMS infrastructure (service provider).

Your task

Application must provide the following functionality:

1. After its startup, create random unique names of goods, that the user (seller) provides, together with random price of each item. This list should be sent as a message via the Offers topic to the other running instances.

   The application should also send the Bank a message requesting account creation. The bank will respond with a message containing the account number.

2. Present to the users (in any trivial form) a list of goods and prices, offered by the other running instances. This list should be updated, whenever a new message from the Offers topic arrives, and any items no longer offered by the sending instance should be removed from the list.

3. Allow the user to choose (again, using any trivial method) goods to buy from another instance. The sale is realized as follows:

   - The buyer sends a message to the seller, containing the name of the requested item and the buyer's account number.
   - The seller sends a message to the buyer, containing the seller's account number, or indicating a refusal to sell (the item was already sold etc.).
   - The buyer sends a message to the Bank, requesting transfer of appropriate amount of money from the buyer's account to the seller's account.
   - The bank sends a message to the seller, notifying the seller of the money transfer from the buyer's account.
The seller removes the item from its list and sends a message to the buyer, confirming the finished sale.

Note: The buyer does not have to offer the item bought for sale.

4. Consider the weaknesses of the communication protocols described above. Extend, document and implement a more robust variant of the protocol (reusing the provided parts is allowed) that will include at least:

- Keeping proper account balances (newly created accounts can have fixed balance for simplicity) and support for account balance queries. (Add a new user command to the client, to show the current balance.)

- More robust sell/buy protocol, considering the buyer's account balance. (Refuse transfer if sender has not enough money.)

- More robust sell/buy protocol, considering that the buyer may transfer less money than the actual price of the required item (assume the price is fixed).

- Better availability of the goods offered by running instances to newly connected instances. (After a new client connects, it should see offers of running without any user action on those clients.)

The messages used to implement the functionality may contain further information that might be needed but not mentioned above. You are to choose the appropriate message types. (For example, the provided Bank implementation uses TextMessage and MapMessage, you are free to choose different types for the communication between the client instances.)

The communication is significantly simplified in 1-3 (e.g. the Bank does not consider account balance), the implementation should however behave reasonably when possible -- refuse to sell an item already requested by a different buyer, etc.

Notes

- 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

- You need two different Session instances
  - The first for asynchronous message handling, the second for synchronous (user-triggered) messages and waiting for their replies
  - A 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

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

Goods offers

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

Buying goods

• Initialize suitable channel for receiving sale requests and create a receiver of its messages
  o Step 2 in the connect() method
• Choose suitable message types for communication between clients
  o MapMessage? ObjectMessage?
• Sending messages requesting a sale
  o Step 1 in the buy() method
• Receiving messages requesting a sale
  o Step 1 in the processSale() method
• Reserve the requested item
  o Step 2 in the processSale() method
• Accept or refuse the sale
  o Step 3 in the processSale() method
• Receive the reply of the sale request message
  o Step 2 in the buy() method
• Money transfer request for the Bank
  o Step 3 in the buy() method (already implemented)
• After receiving the transaction notification from the Bank (implemented), send a finished sale confirmation
  o Step 3 in the processBankReport() method
• Receive the confirmation, notify the user
  o Step 4 in the buy() method
Common mistakes

- Using a wrong JMS session
- No synchronization
- Inability to buy goods after the previous attempt fails
  - When the buyer had not enough money
  - Make sure to return the goods to available state
- No handling of "exceptional" cases
  - For example entering wrong client name or goods name

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)