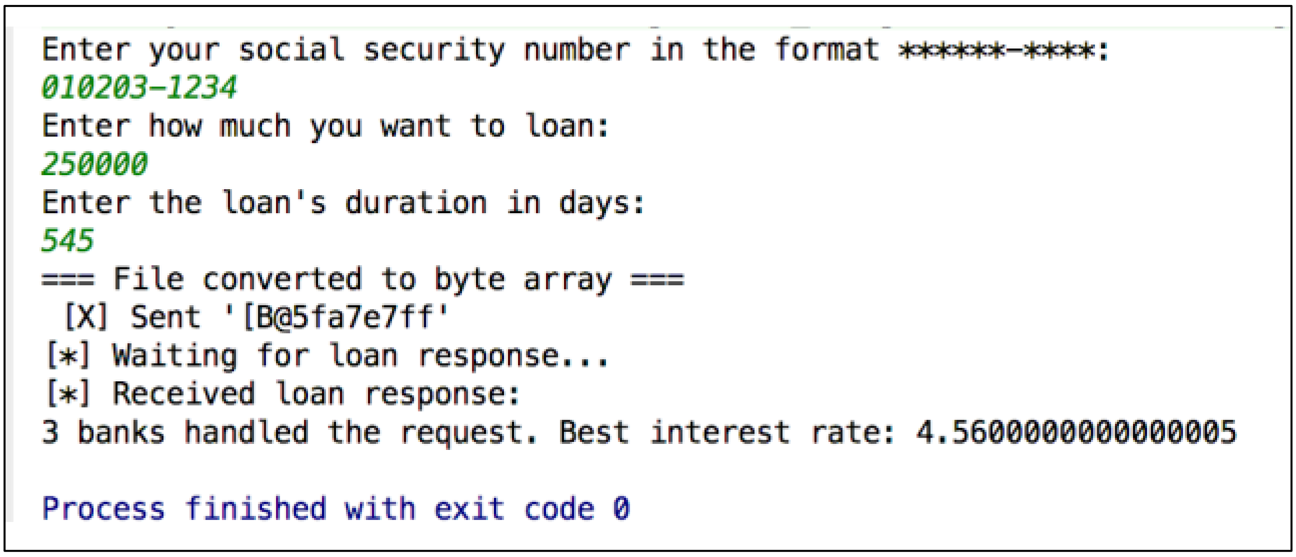
Process flow

Loan Broker – Group 14

‘Loan Request’ component

Output



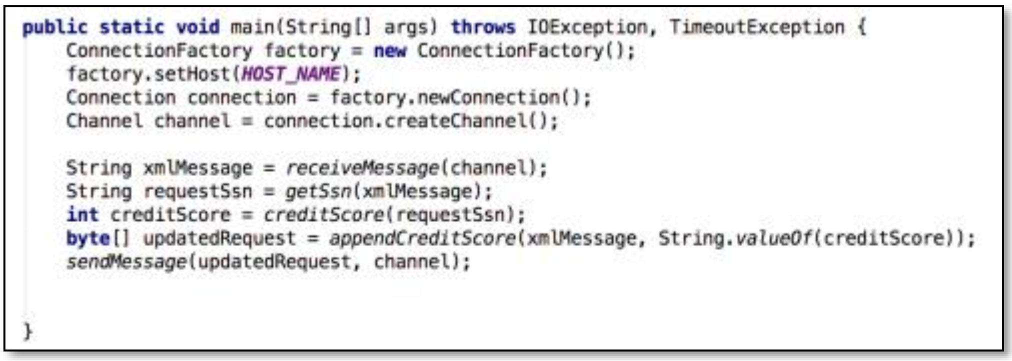
```
Enter your social security number in the format ******-****:
010203-1234
Enter how much you want to loan:
250000
Enter the loan's duration in days:
545
=== File converted to byte array ===
 [X] Sent '[B@5fa7e7ff'
[*] Waiting for loan response...
[*] Received loan response:
3 banks handled the request. Best interest rate: 4.5600000000000005

Process finished with exit code 0
```

The user requests a loan from the Loan Request component by inputting his/her social security number, desired loan amount and the loan duration.
Afterwards, the request will be converted to an XML string, converted to bytes and put on a queue for the next component to pick it up. From there, the message will be processed through all the remaining components, until it finally arrives at the Loan Request component again, which then displays the best possible interest rate.

‘Get Credit Score’ component

Content enricher

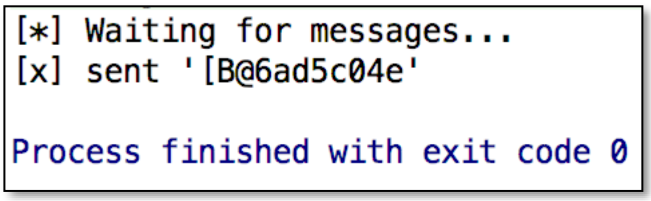
Code



```java
public static void main(String[] args) throws IOException, TimeoutException {
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost(HOST_NAME);
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    String xmlMessage = receiveMessage(channel);
    String requestSsn = getSsn(xmlMessage);
    int creditScore = creditScore(requestSsn);
    byte[] updatedRequest = appendCreditScore(xmlMessage, String.valueOf(creditScore));
    sendMessage(updatedRequest, channel);


}
```

This component requests a credit score from an external web service, based on the received message with the users input social security number. Afterwards it appends the XML string with the credit score, and puts it on the next queue.

Output



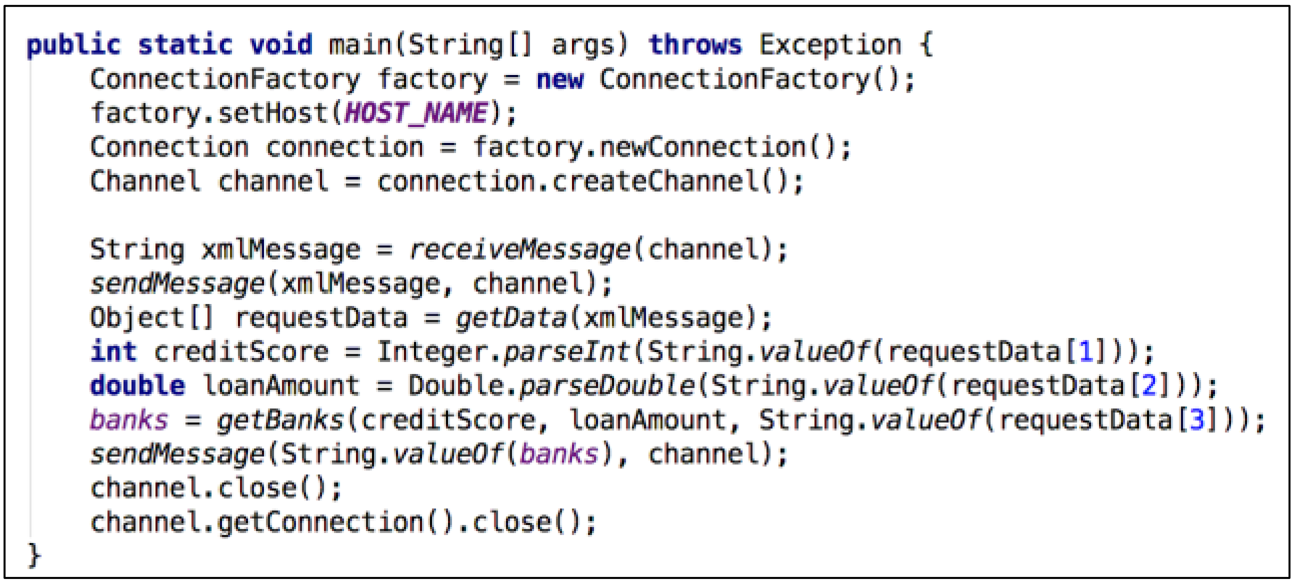
```
[*] Waiting for messages...
[x] sent '[B@6ad5c04e'

Process finished with exit code 0
```

Very simple output, indicating that the message has been successfully processed and put on the next queue.

‘Get Banks’ component

Content enricher

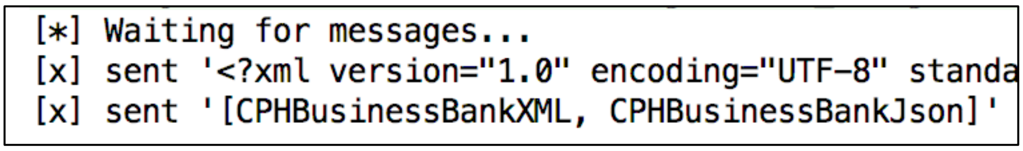
Code



```java
public static void main(String[] args) throws Exception {
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost(HOST_NAME);
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    String xmlMessage = receiveMessage(channel);
    sendMessage(xmlMessage, channel);
    Object[] requestData = getData(xmlMessage);
    int creditScore = Integer.parseInt(String.valueOf(requestData[1]));
    double loanAmount = Double.parseDouble(String.valueOf(requestData[2]));
    banks = getBanks(creditScore, loanAmount, String.valueOf(requestData[3]));
    sendMessage(String.valueOf(banks), channel);
    channel.close();
    channel.getConnection().close();
}
```

This component requests a list of the banks that are interested in providing a loan, based on the acquired credit score. That’s why this code extracts the credit score and the loan amount, makes an inquiry to the banks using the *getBanks*-method.
Afterwards, the loan request and the list of applicable banks are put on the next queue.

Output



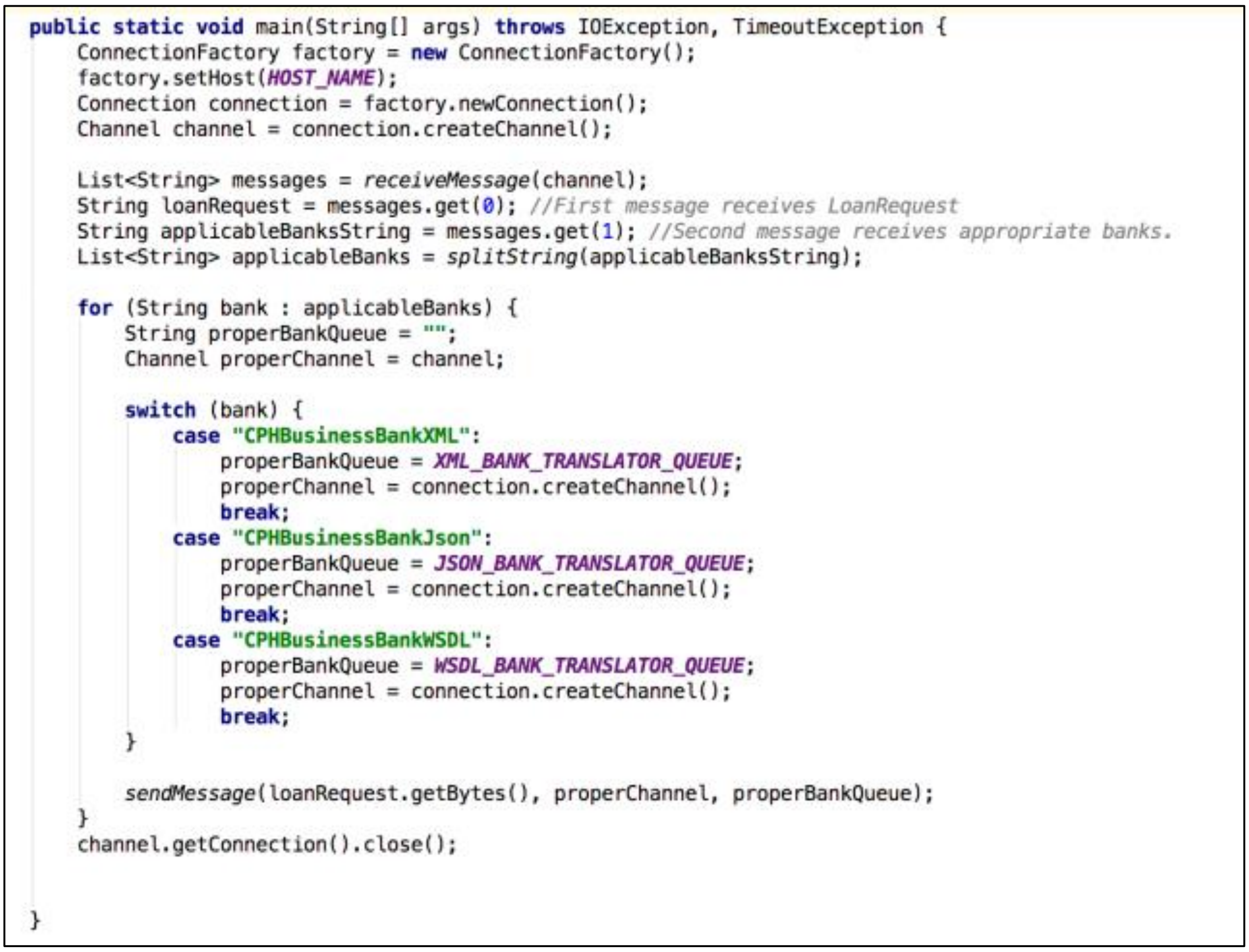
```
[*] Waiting for messages...
[x] sent '<?xml version="1.0" encoding="UTF-8" standa
[x] sent '[CPHBusinessBankXML, CPHBusinessBankJson]'
```

Simple output indicating that the loan request is forwarded along with the applicable banks.

‘Recipient list’ component

Recipient list

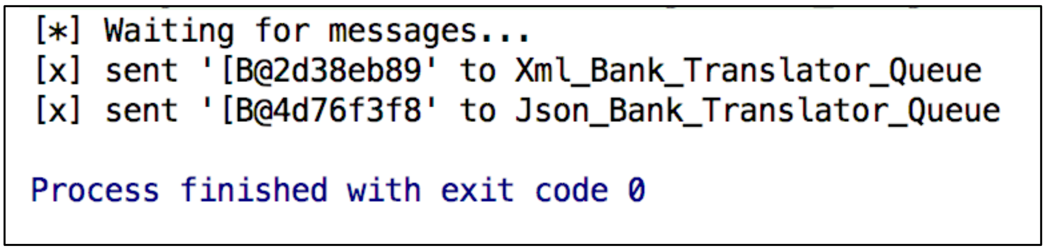
Code



```
public static void main(String[] args) throws IOException, TimeoutException {
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost(HOST_NAME);
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    List<String> messages = receiveMessage(channel);
    String loanRequest = messages.get(0); //First message receives LoanRequest
    String applicableBanksString = messages.get(1); //Second message receives appropriate banks.
    List<String> applicableBanks = splitString(applicableBanksString);

    for (String bank : applicableBanks) {
        String properBankQueue = "";
        Channel properChannel = channel;

        switch (bank) {
            case "CPHBusinessBankXML":
                properBankQueue = XML_BANK_TRANSLATOR_QUEUE;
                properChannel = connection.createChannel();
                break;
            case "CPHBusinessBankJson":
                properBankQueue = JSON_BANK_TRANSLATOR_QUEUE;
                properChannel = connection.createChannel();
                break;
            case "CPHBusinessBankWSDL":
                properBankQueue = WSDL_BANK_TRANSLATOR_QUEUE;
                properChannel = connection.createChannel();
                break;
        }

        sendMessage(loanRequest.getBytes(), properChannel, properBankQueue);
    }
    channel.getConnection().close();


}
```

The recipient list inspects the list of banks that are applicable for providing a loan, and puts them on a proper queue, so the correct translators will receive the request.
This is necessary due to the fact that each separate Translator has got a specific queue that it subscribes to.

Output



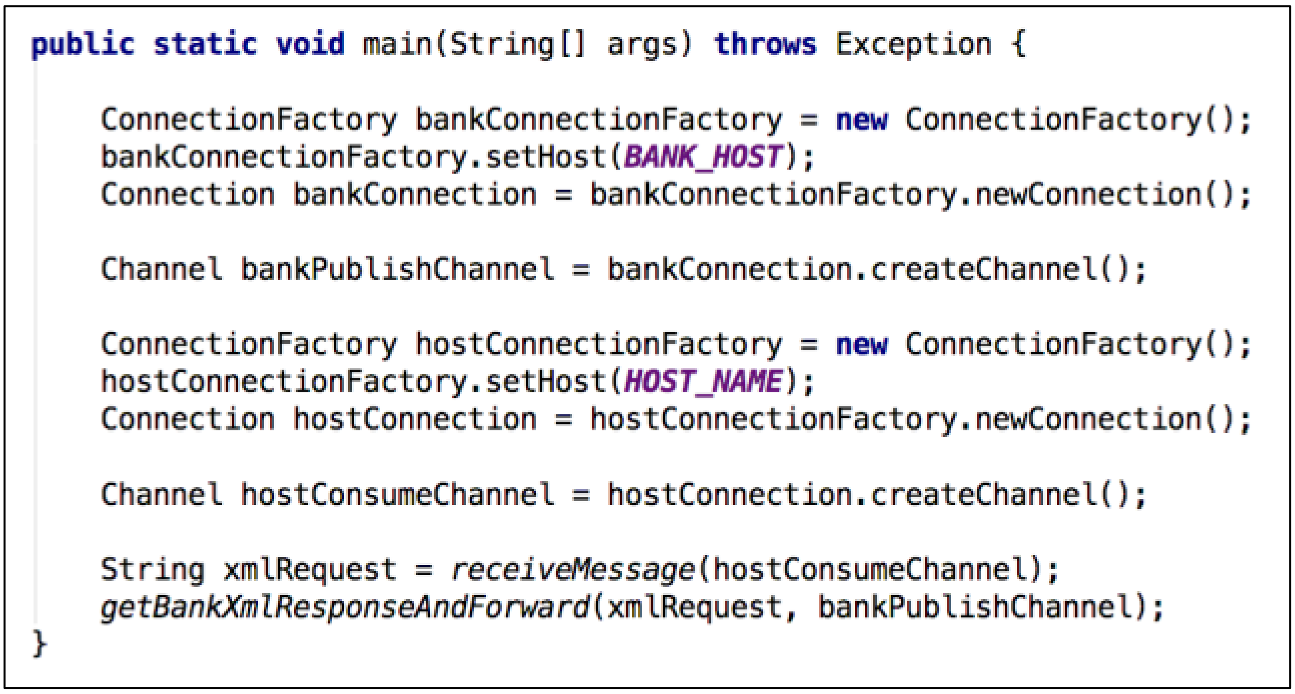
```
 [*] Waiting for messages...
 [x] sent '[B@2d38eb89' to Xml_Bank_Translator_Queue
 [x] sent '[B@4d76f3f8' to Json_Bank_Translator_Queue

Process finished with exit code 0
```

A simple output message indicating that the forwarding was successful, and which banks the request is being sent to.

‘XML Translator’ component

Translator

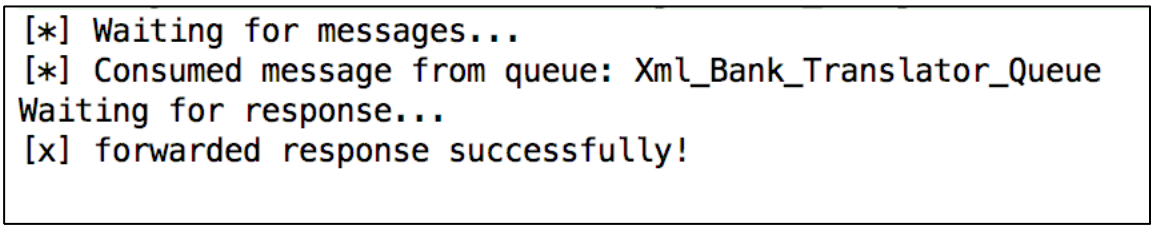
Code



```java
public static void main(String[] args) throws Exception {

    ConnectionFactory bankConnectionFactory = new ConnectionFactory();
    bankConnectionFactory.setHost(BANK_HOST);
    Connection bankConnection = bankConnectionFactory.newConnection();

    Channel bankPublishChannel = bankConnection.createChannel();

    ConnectionFactory hostConnectionFactory = new ConnectionFactory();
    hostConnectionFactory.setHost(HOST_NAME);
    Connection hostConnection = hostConnectionFactory.newConnection();

    Channel hostConsumeChannel = hostConnection.createChannel();

    String xmlRequest = receiveMessage(hostConsumeChannel);
    getBankXmlResponseAndForward(xmlRequest, bankPublishChannel);
}
```

The XML translator doesn’t actually have to do a lot of translating since the incoming request is already in XML format. Therefore, the XML Translator component is just in charge of forwarding the request to the XML bank, and supplying the correct reply queue.

Output



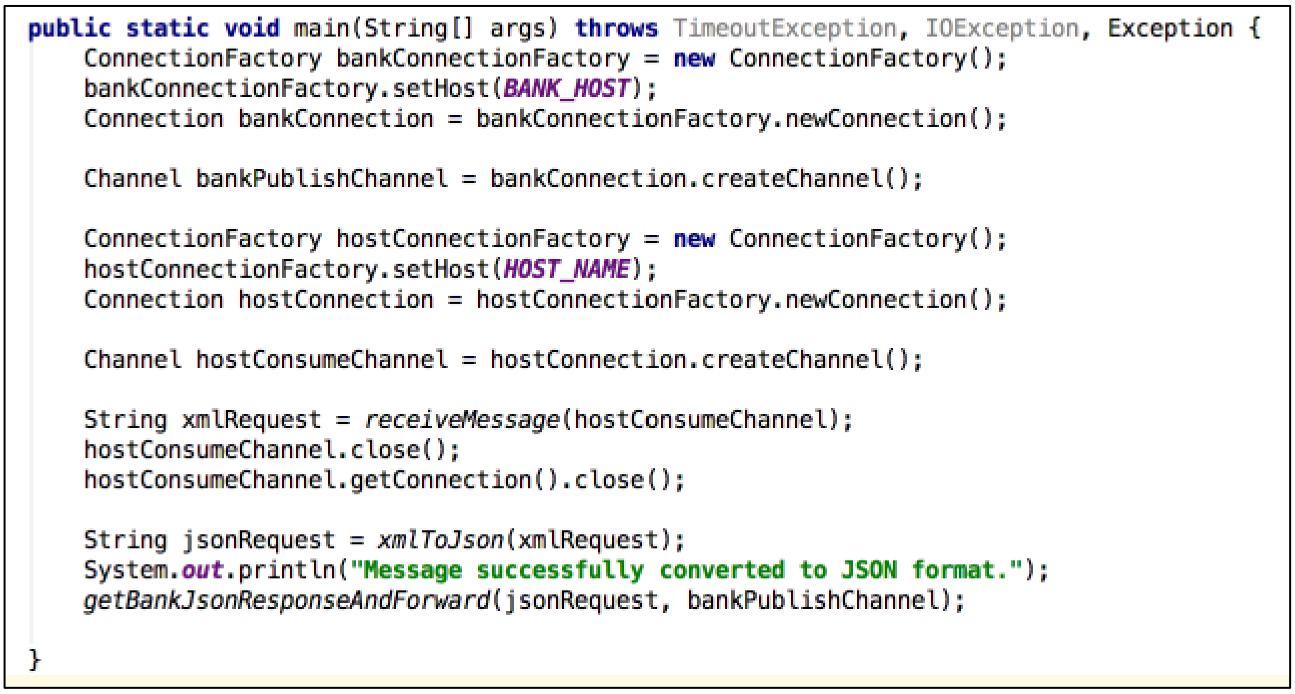
```
 [*] Waiting for messages...
 [*] Consumed message from queue: Xml_Bank_Translator_Queue
Waiting for response...
 [x] forwarded response successfully!
```

A simple output message indicating that the request has successfully been forwarded to the correct bank.

‘JSON Translator’ component

Translator

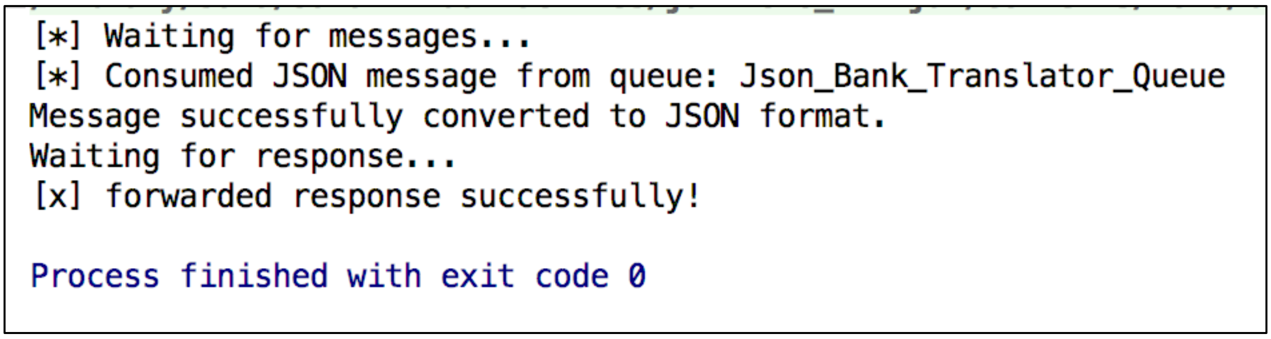
Code



```java
public static void main(String[] args) throws TimeoutException, IOException, Exception {
    ConnectionFactory bankConnectionFactory = new ConnectionFactory();
    bankConnectionFactory.setHost(BANK_HOST);
    Connection bankConnection = bankConnectionFactory.newConnection();

    Channel bankPublishChannel = bankConnection.createChannel();

    ConnectionFactory hostConnectionFactory = new ConnectionFactory();
    hostConnectionFactory.setHost(HOST_NAME);
    Connection hostConnection = hostConnectionFactory.newConnection();

    Channel hostConsumeChannel = hostConnection.createChannel();

    String xmlRequest = receiveMessage(hostConsumeChannel);
    hostConsumeChannel.close();
    hostConsumeChannel.getConnection().close();

    String jsonRequest = xmlToJson(xmlRequest);
    System.out.println("Message successfully converted to JSON format.");
    getBankJsonResponseAndForward(jsonRequest, bankPublishChannel);

}
```

The JSON translator is little different, because this time it is necessary to convert the incoming XML request to a proper JSON format that the JSON bank can read and progress. After the JSON Translator component has converted the message, it forwards the newly formatted JSON request the JSON bank, that puts it on the correct reply queue.

Output

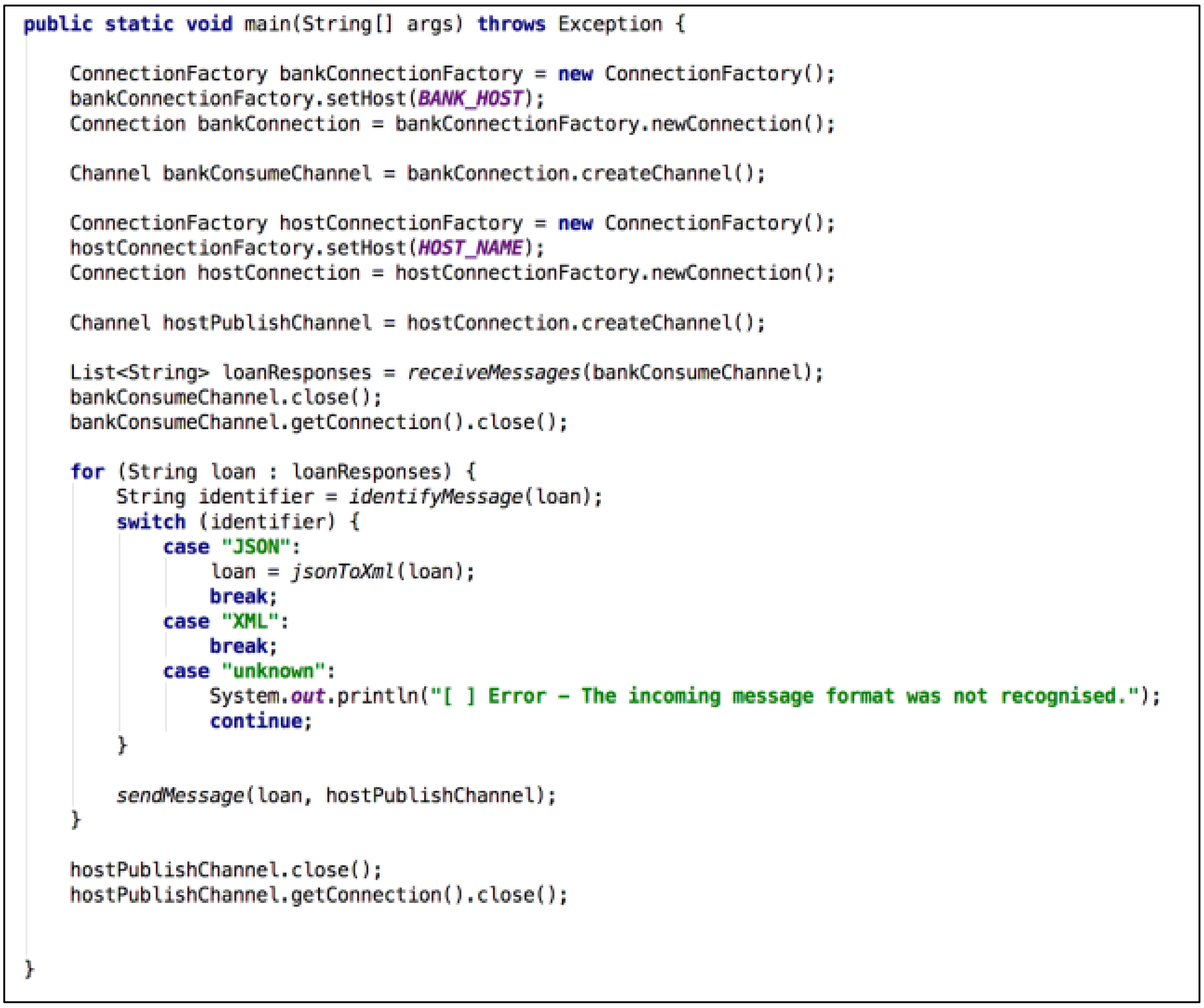


```
 [*] Waiting for messages...
 [*] Consumed JSON message from queue: Json_Bank_Translator_Queue
Message successfully converted to JSON format.
Waiting for response...
 [x] forwarded response successfully!

Process finished with exit code 0
```

A simple output message indicating that the original XML request has been received, converted and forwarded successfully to the JSON bank.

‘Normalizer’ component

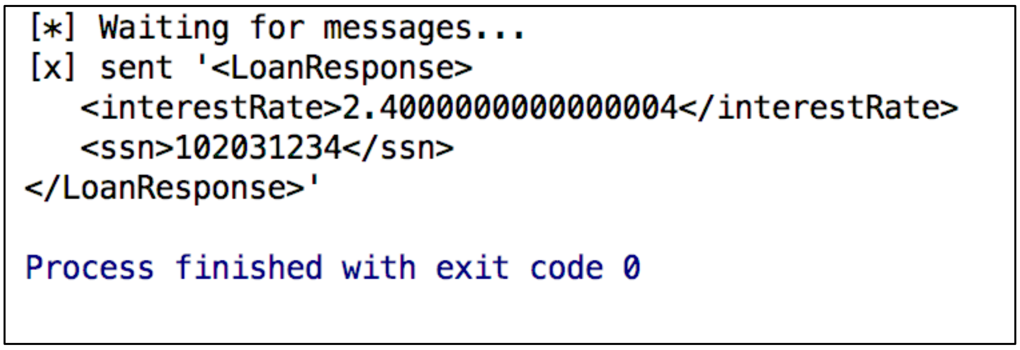
Code



```java
public static void main(String[] args) throws Exception {

    ConnectionFactory bankConnectionFactory = new ConnectionFactory();
    bankConnectionFactory.setHost(BANK_HOST);
    Connection bankConnection = bankConnectionFactory.newConnection();

    Channel bankConsumeChannel = bankConnection.createChannel();

    ConnectionFactory hostConnectionFactory = new ConnectionFactory();
    hostConnectionFactory.setHost(HOST_NAME);
    Connection hostConnection = hostConnectionFactory.newConnection();

    Channel hostPublishChannel = hostConnection.createChannel();

    List<String> loanResponses = receiveMessages(bankConsumeChannel);
    bankConsumeChannel.close();
    bankConsumeChannel.getConnection().close();

    for (String loan : loanResponses) {
        String identifier = identifyMessage(loan);
        switch (identifier) {
            case "JSON":
                loan = jsonToXml(loan);
                break;
            case "XML":
                break;
            case "unknown":
                System.out.println("[ ] Error - The incoming message format was not recognised.");
                continue;
        }

        sendMessage(loan, hostPublishChannel);
    }

    hostPublishChannel.close();
    hostPublishChannel.getConnection().close();


}
```

The Normalizer components job is to make sure that every separate response from each bank is forwarded as a single, readable format. In this case, we have chosen XML. Therefore, the Normalizer has to identify the incoming loan response. If it’s a JSON format, convert it to XML. If it’s already XML format, it does nothing.
Finally, the Normalizer forwards the message to the last component.

Output

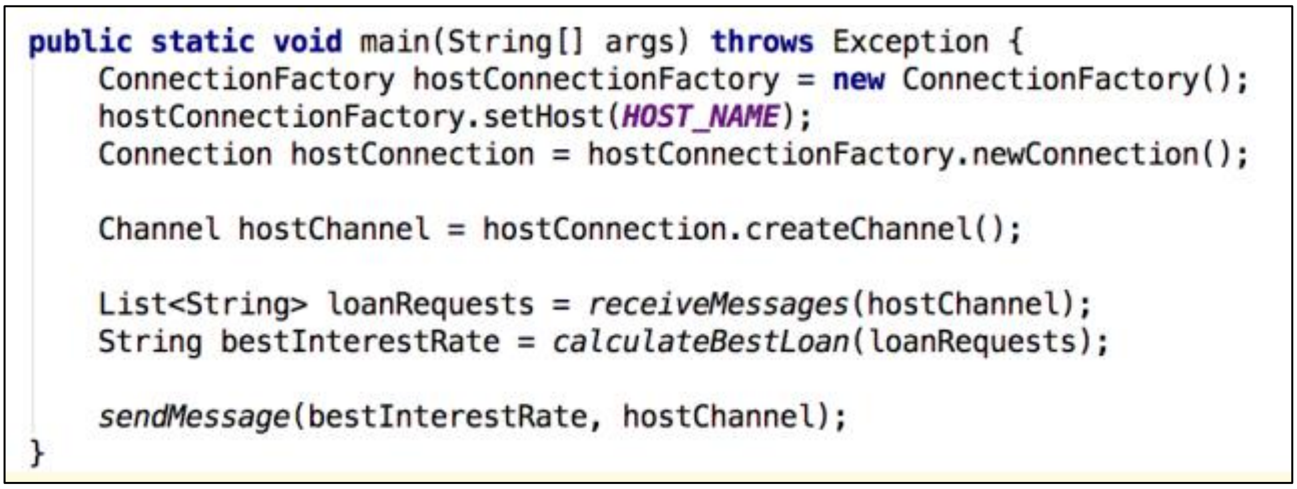


```
[*] Waiting for messages...
[x] sent '<LoanResponse>
   <interestRate>2.4000000000000004</interestRate>
   <ssn>102031234</ssn>
</LoanResponse>'

Process finished with exit code 0
```

A simple output, indicating the success and content of the message being forwarded to the next component.

‘Aggregator’ component

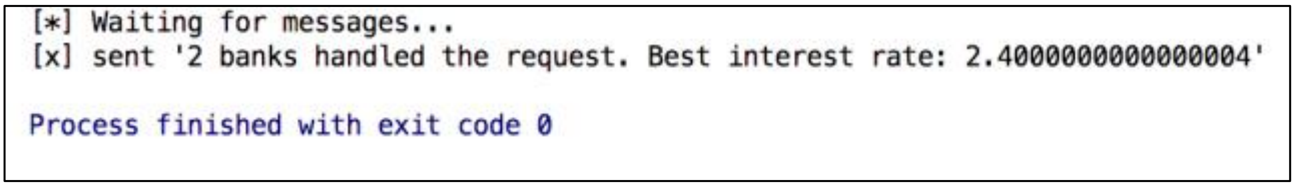
Code



```java
public static void main(String[] args) throws Exception {
    ConnectionFactory hostConnectionFactory = new ConnectionFactory();
    hostConnectionFactory.setHost(HOST_NAME);
    Connection hostConnection = hostConnectionFactory.newConnection();

    Channel hostChannel = hostConnection.createChannel();

    List<String> loanRequests = receiveMessages(hostChannel);
    String bestInterestRate = calculateBestLoan(loanRequests);

    sendMessage(bestInterestRate, hostChannel);
}
```

The Aggregator is the last component in the Loan Broker chain, and its job is to collect all the single-formatted messages from the Normalizer, and analyze them. In this case, it finds the best interest rate provided by the banks, among the different loan responses. Afterwards, it puts the best interest rate on a queue, in which the ‘Loan Request’ component picks up, and thus the Loan Broker circle is complete.

Output



```
[*] Waiting for messages...
[x] sent '2 banks handled the request. Best interest rate: 2.4000000000000004'

Process finished with exit code 0
```

A simple output, indicating the final message being sent back to the Loan Broker.