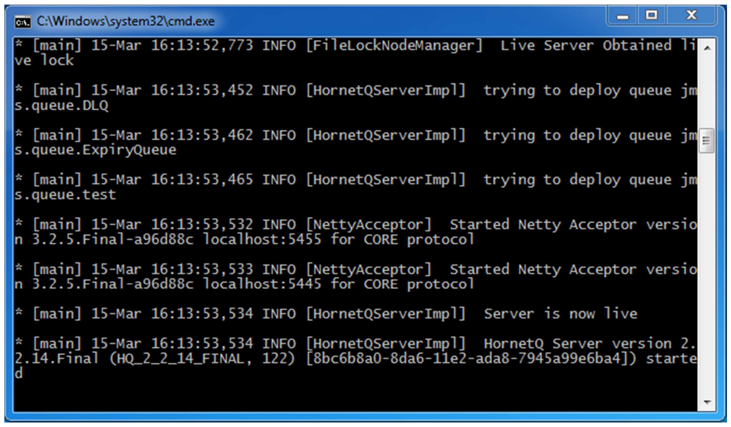
Module 24: Spring JMS

Exercise 24.1 – Spring JMS Calculator

The Setup:

In this exercise we will build a JMS-based calculator, somewhat similar to the RMI-based calculator. Just like the RMI exercises, this exercise will also have two projects, a JMS sender and a JMS receiver.

We will be using HornetQ (download from \\CS5\Public\Courses\CS544\Salek\Software) as our JMS server, you can start HornetQ by double clicking on **..\hornetq-2.2.14.Final\bin\run.bat**, which should open the following window:



```
* [main] 15-Mar 16:13:52,773 INFO [FileLockNodeManager]  Live Server Obtained li
ve lock

* [main] 15-Mar 16:13:53,452 INFO [HornetQServerImpl]  trying to deploy queue jm
s.queue.DLQ

* [main] 15-Mar 16:13:53,462 INFO [HornetQServerImpl]  trying to deploy queue jm
s.queue.ExpiryQueue

* [main] 15-Mar 16:13:53,465 INFO [HornetQServerImpl]  trying to deploy queue jm
s.queue.test

* [main] 15-Mar 16:13:53,532 INFO [NettyAcceptor]  Started Netty Acceptor versio
n 3.2.5.Final-a96d88c localhost:5455 for CORE protocol

* [main] 15-Mar 16:13:53,533 INFO [NettyAcceptor]  Started Netty Acceptor versio
n 3.2.5.Final-a96d88c localhost:5445 for CORE protocol

* [main] 15-Mar 16:13:53,534 INFO [HornetQServerImpl]  Server is now live

* [main] 15-Mar 16:13:53,534 INFO [HornetQServerImpl]  HornetQ Server version 2.
2.14.Final (HQ_2_2_14_FINAL, 122) [8bc6b8a0-8da6-11e2-ada8-7945a99e6ba4]) starte
d
```

Now that we have started the JMS server, we can continue by opening the projects. Open both the **exercise24_1-sender** and the **exercise24_1-receiver** projects. In addition to the regular log4j and spring context dependencies both projects also have the following dependencies:

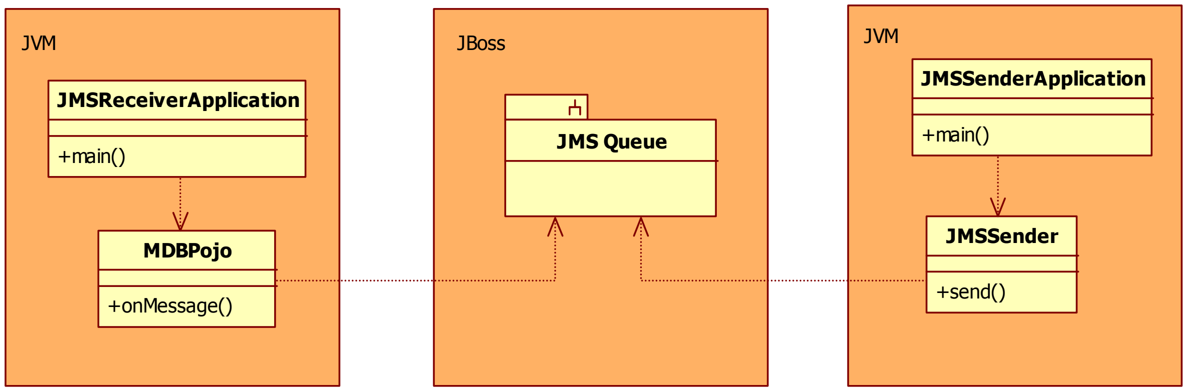
```xml
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jms</artifactId>
    <version>4.0.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.hornetq</groupId>
    <artifactId>hornet-jms-client</artifactId>
    <version>2.4.1.Final</version>
</dependency>
<dependency>
    <groupId>jboss</groupId>
    <artifactId>jnp-client</artifactId>
    <version>4.2.2.GA</version>
</dependency>
```

The receiver project depends on the sender project, so be sure to add the sender project to the receivers' dependencies, see exercise23 for details on how to setup inter project dependencies.

The Application:

The provided projects have a JMS sender and a JMS receiver application. The JMS receiver application starts a MDBPojo message bean that listens and waits until a message arrives in the JMS Queue. The JMS Sender application uses the JMS Sender object to send a new message containing a Person object to the Queue.

Once the message has been put in the Queue, the MDBPojo retrieves it, takes out the Person object, and writes to the console: Hello : "Hello" followed by 'firstName' and 'lastName' .



To execute the example code, first start **JMSReceiverApp.java**. This should give the output:

```
JMS receiver is running ...
```

Then start the **JMSSenderAppl.java,** which should output:

```
Sending message with person object : John Doe
```

After which Eclipse will switch back to the JSMReceiverApplications to show:

```
JMS receiver is running ...
MDBPojo receives message with person object : John Doe
```

Important: the Sender uses a ConfigurableApplicationContext, so we can close it, and thereby close the connection to the JMS server. Not doing so can cause a delay in JMS message delivery.

The Exercise:

Study the code, and once you are comfortable with it, write a JMS calculator application where the JMS receiver implements a stateful calculator that receives commands by means of JMS messages. Have the sender output the commands it is about to send, and have the receiver output the resulting calculations.

Again, because our calculator is stateful and JMS messages can be handled by multiple threads, it is important to make the **onMessage()** method of the calculator **synchronized**.