Enterprise Systems & Architecture

Lab 9 (Week 10): Message Oriented Middleware using JMS

Note: You can do this lab sheet over this week and next week.

In this lab we will be installing a JEE Application server and using some of its *Message Oriented Middleware* capabilities. Specifically we want to utilise its JMS messaging broker – we will be covering this in class but you can find out more here.

JBoss – Download, Extract and Run

- Firstly we will download and install the JBoss Java Enterprise Edition (JEE) Application Server and start it up. We will use *JBoss 4.2.3*
- Goto:

http://sourceforge.net/projects/jboss/files/JBoss/JBoss-4.2.3.GA/

- Choose the download link for: **jboss-4.2.3.GA.zip**
- Extract the zip file to a suitable location on disk.
- Open a command prompt and change directory to your JBoss installation folder and type (there is a script for linux also):

./bin/run.bat

Exercise 1: Eclipse – Create a Java Project and Write an example Messaging Client Program

- Start Eclipse
- Create a new java project: *File -> New -> Java Project*
- Give the project a name *HelloWorldMessage* and click *Finish*
- Right-Click on the project name:
    - *New -> Class*
    - leave package blank
    - name of class: *HelloWorldMessage*
- Right-Click on the Project Name
    - *New Folder* -> name it "*lib*"
    - Using File Explorer on your PC - Copy
        - <JBOSS_HOME>\server\default\lib\jboss-j2ee.jar to *lib* directory
        - <JBOSS_HOME>\client\jbosall-client.jar to *lib* directory
        - Note: < JBOSS_HOME> is your JBoss installation folder.
- Right-Click on the Project Name -> *Build Path -> Configure Build Path -> Libraries - > Add Jars*
    - Add the jars in the lib folder to the build path
- Refresh the project (right-click on project -> *Refresh*)
- Copy example JMS client code from Brightspace (HelloWorldMessage.java) to the new class and save (just overwrite the skeleton code already there)
- Right-Click on java class -> *Run As -> Java Application*
- **Have a look through the code to see what each line of code is doing…**

An Asynchronous JMS Receiver Client

In the above messaging exercise you implemented a java class that was both a sender and receiver of messages. The receiving code was a *Synchronous* client – it connected to the broker and requested any messages.

Here we want to write an *Asynchronous* receiving client.

Exercise 2

Write an Asynchronous Listener

- In the same java project that you used in the last exercise, create a new java class called *MyListener*. This class needs to implement the *MessageListener* java interface.
- Add the following method to your class:

```
public void onMessage(Message message) {
        TextMessage msg = (TextMessage) message;
        try {
                System.out.println("Received By Listener: " + msg.getText());
        } catch (JMSException ex) {
                ex.printStackTrace();
        }
}
```

- An instance of this class will be created in our application code and will act as our asynchronous listener for messages.

Write a new Message sender and receiver

- Make a copy of the *HelloWorldMessage* class from the previous exercise.
- Firstly, comment out the 4 lines of code at step 7.
- In the existing code, at step 10, the following line synchronously reads messages from the queue:

  Message msg = myMsgConsumer.receive():

- We need to replace this with:

  myMsgConsumer.setMessageListener(**new** MyListener());

- Add the following code to your class (replacing the 4 lines of code at step 11) so that it sends messages over a period of 17 seconds, in doing this we can test our MyListener code.

```
//Wait for messages
System.out.print("Waiting for messages\n\n");
for (int i = 0; i < 17; i++) {
        Thread.sleep(1000);
        System.out.print(".");

        //Every 4 seconds, put a message on the queue
        TextMessage myTextMsg = null;
        if (i % 4 == 0){
                myTextMsg = mySess.createTextMessage();
                myTextMsg.setText("Hello World: Time Elapsed has been: " + i + " seconds." );

                System.out.println("\nSending Message: " + myTextMsg.getText());
                myMsgProducer.send(myTextMsg);
        }

}
System.out.println();
```

- Run your messaging code.
- Do you notice anything about the output? Run the code again.

Optional - Exercise 3: Sending XML Messages.

- In your code, change the messages that you are sending so that they are XML messages (text messages with the text formatted as XML) – use our original note.xml example

```
<note>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

- Change the code in the onMessage() method of the listener so that it retrieves the data from the message and outputs the following to the console:

Received a note: Don't forget me this weekend!
Heading: Reminder
From: Jani
To: Tove
Message: Reminder