Message Oriented Middleware with JMS on GlassFish

This homework has three tasks. Task one consists of a short tutorial on writing and running a JMS application using Netbeans and GlassFish. You should not submit task one.

Task two consists of a small JMS exercise of adding a second queue to the tutorial, and reading from it with a second servlet.

Task three expands on our past AlFlickrBet project and rewrites it using JMS Topics and Temporary Queues. The resulting AlFlickrBetDeluxe application is easily extensible because of the message-based architecture, allowing for adding (or taking away) additional categories of word and picture associations without affecting the rest of the application.

Task One: Tutorial

GlassFish comes with the JMS Provider, Sun Message Queue. First, using the GlassFish administrative server, we need to administratively establish a ConnectionFactory and a Queue. JNDI (the Java Naming and Directory Interface) will be used to get access to these administrated objects.

Messages coming out of the Queue may be read synchronously with a receive method call or asynchronously by implementing a message listener. A message driven bean may also be used to handle incoming, asynchronous messages.

Set up a ConnectionFactory and Queue

1) Run Netbeans
2) Choose Services/Servers/Right Click and start GlassFish v3 Domain.
3) Right-click again on GlassFish v3 Domain and choose “View Admin Console”
   a) The default login is “admin”
   b) The default password is “adminadmin”
   c) Within the admin console, expand Resources/JMS Resources.
   d) Select Connection Factories
      i) Select New from the menu.
      ii) Enter the Pool Name:jms/myCoolConnectionFactory.
      iii) From the drop down list, select the type javax.jms.ConnectionFactory
      iv) Click OK.
   e) Under JMS Resources, select Destination Resources.
      i) Select New from the menu.
      ii) Enter the JNDI Name:jms/myCoolQueue.
      iii) Enter the Physical Destination Name:myCoolQueue.
      iv) From the drop down list, select the type javax.jms.Queue
      v) Click OK.
   f) You can now logout of the admin console.
Build an application – a web component and a Message Driven bean

4) Return to Netbeans and choose Projects.
5) Select File/New Project
   a) Select Java EE and Enterprise Application.
   b) The project name is MyCoolJEEProject.
   c) Create an EJB Module and a Web Application Module.
   d) Click finish.
6) Populate the EJB module with a Message Driven Bean.
   a) From the Project View, Right Click MyCoolJEEProject-ejb.
   b) Select New Message Driven Bean.
      - The EJB Name is MyCoolMDB and the package name is mycoolmdb
   c) Select the server destination as jms/myCoolQueue.
      - Notice that this is the queue that you created earlier.
   d) Select Finish and you should see a default Message Driven Bean.
7) Replace the MyCoolMDB code with the code found in:
   http://andrew.cmu.edu/course/95-702/homework/Fall2010/Fall2010Project6/MyCoolMDB.java
8) Build a web application that sends messages to the queue.
   a) In the Project View, expand the MyCoolJEEProject-war.
   b) Expand Web Pages and double click index.jsp.
   c) Replace index.jsp with the code found in:
      http://andrew.cmu.edu/course/95-702/homework/Fall2010/Fall2010Project6/index.jsp
9) Create servlet to collect the text from the browser and deliver it to the Message Driven Bean.
   a) In the Project View, select MyCoolJEEProject-war.
   b) Right click and choose New Servlet.
   c) The servlet name is MyCoolServlet.
   d) The package name is mycool servlet.
   e) Choose Next through to Finish.
   f) Replace MyCoolServlet.java with the code found in:
      http://andrew.cmu.edu/course/95-702/homework/Fall2010/Fall2010Project6/MyCoolServlet.java
10) From the Project View, right click MyCoolJEEProject and select deploy.
11) Run a browser and visit http://localhost:8080/MyCoolJEEProject-war/

Nothing needs to be turned in for Part One

Task Two – Message Queues

1) As demonstrated in class, modify the tutorial task such that MyCoolMDB adds text to the message, and sends the message to a new Queue named responseQueue.
2) Write a new servlet, FetchResponses that reads all available messages in the responseQueue and displays them on a web page.
   a) If no messages are available, the servlet should clearly state that on the response page.
   b) If there are one or more messages in the Queue, all should be displayed.
3) Provide screen shots of your web pages.
Task Three – AlFlickerBetDeluxe – Topics and Temporary Queues

This task will be demonstrated in class. You are to build upon Project 2 Task 6, and using JMS Topics and Temporary Queues, have multiple agents searching for pictures that match a given letter.

For example, you already reply to the letter "a" with something like "a is for alligator" and provide a picture of an alligator. Your task is to add an additional search in another category, such as food, trees, cities, or another interesting category of your choice, and provide an additional reply such as "a is for apple", "a is for aspen", or "a is for Albuquerque".

You should be able to reuse some of the code from your former project, including:
1) The mapping of letters to words
2) The Flickr search
3) The parsing of Flickr’s XML response
4) Crafting of the picture URL from components in the Flickr response

And replicating this functionality to searching another category should be trivial.

The overall architecture of your application, however, should be quite different.
1) The application will use a Topic. Be sure to create it first in the GlassFish Admin Console as demonstrated in class.
2) The user experience should begin with a page (jsp) that introduces the game and asks the user for a letter.
3) A servlet should then:
   a) Do the setup necessary to publish to the Topic
   b) Create a Temporary Queue for the response from the category search agents to reply to.
   c) Add the Temporary Queue to the Message being published to the Topic
      i) E.g. Queue replyQueue = qsession.createTemporaryQueue();
          msg.setJMSReplyTo(replyQueue);
   d) Put the user’s letter onto the Message
   e) Publish the message to the Topic
   f) Receive response messages from the Temporary Queue
      i) Note, there could zero or more responses. Be sure to wait long enough in your "receive"
         to allow for a response. If the agent found a picture, it should reply with a message. If it did not, it should not send a message.
      ii) Make a suitable reply to the user with the words and pictures corresponding to their letter.

Note: A Temporary Queue is an alternative to setting up a permanent Queue in the Admin Console. Its lifetime is the life of the JMS session that it is created in. For our application, a permanent response Queue would not be suitable, for the responses need to be sent to potentially multiple simultaneous users. If two players were playing at the same time, and one chose "a" and the other "b" their responses being returned in the same Queue could be mixed up. By using Temporary Queues created in each servlet instance, tied to a JMS session in that servlet, the responses will come back to the right servlet for each user.

4) Create one Message Driven Bean that is subscribed to the Topic
   a) In the onMessage method, get the letter from the message
   b) Map the letter to the name of an animal
   c) Search Flickr for a picture of the animal
d) Parse the resulting XML

e) Craft the picture URL from the components of the Flickr response.
f) Get the replyTo Temporary Queue from the message, and do the setup necessary to make a response.
g) Craft a message to respond with. I recommend using a MapMessage with attributes of "word" and "pictureURL".
h) Send the message to the Temporary Queue.

At this point you should be able to test the application and it should work similarly to your original application.

5) Create a second Message Driven Bean that also subscribes to the same Topic
   a) Copy, and modify your first MDB's code to map the letter to a word in a different category and search Flickr.
   b) Otherwise, your code should be the same, and should reply to the same Temporary Queue that is passed in the Message with the letter.

6) Provide browser screen shots of your web application, including cases:
   a) Initial screen
   b) Screens with two (or more, if you like) responses.
   c) Screens with only one response (e.g. u is for unicorn returned nothing, but there is a picture of u is for udon)
   d) Screens with no responses (e.g. for the letter q) (If you don't have any letter with no responses, state that with the screen shots.)

Note: If you get the message: WARNING: MQJMSRA_TD4001: delete():Cannot delete TemporaryDestination with active consumers", this is apparently a bug in the GlassFish Message Oriented Middleware, so you can ignore it.

Summary of what to submit to Blackboard:

1) Task One: Nothing to submit
2) Task Two:
   a) MyCoolJeeProject NetBeans project directory. It should contain the –ejb and –war subdirectories.
   b) Screen shots
3) Task Three:
   a) AlFlickerBetDeluxe project directory (with included –ejb and –war directories).
   b) Screen shots
4) Complete the Project 6 reflection questions on Blackboard
Grading:

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tutorial</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Message Queues</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>AlFlickerBetDeluxe Topics and Temporary Queues</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>Reflection questions</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>