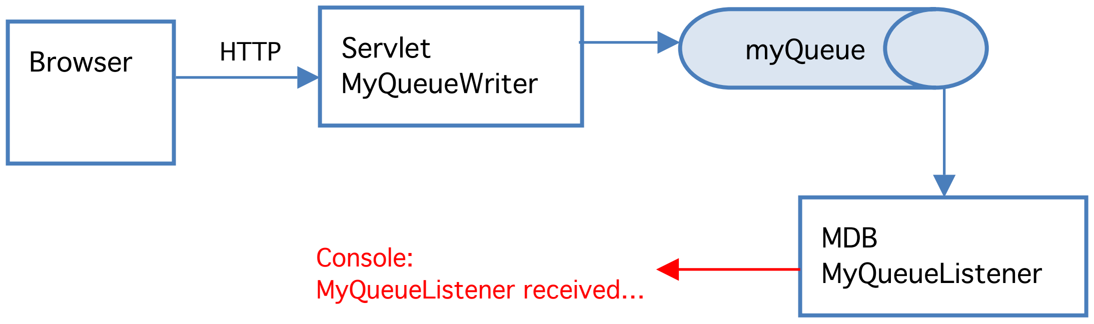
95-702 Distributed Systems
Lab 9 – JMS

Task 1 – Single Queue



GlassFish comes with the JMS Provider, Open Message Queue (OpenMQ). 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.

Set up a ConnectionFactory and Queue

1) Run Netbeans
2) Choose Services, expand Servers, right click on GlassFish Server 4 and "start".
    a) If the Services tab is not shown, make it visible by choosing Services under the Window menu.
3) Once GlassFish is running (it takes a minute or more), right-click again on GlassFish Server and choose "View Domain Admin Console"
    a) You will be prompted to log in if you chose to require an administration password at the time GlassFish Server was installed.
    b) Within the admin console, expand Resources then JMS Resources.
    c) Select Connection Factories
        i) Select New from the menu.
        ii) Enter the JNDI Name: jms/myConnectionFactory.
        - It is a convention to name JMS resources as jms/someResourceName
        iii) From the drop down list, select the type javax.jms.ConnectionFactory
        iv) Leave the other defaults as they are and click OK.
    d) Under JMS Resources, select Destination Resources.
        i) Select New from the menu.
        ii) Enter the JNDI Name: jms/myQueue.
        iii) Enter the Physical Destination Name: myQueue.
        - The JNDI name is the string used to lookup the physical destination queue in the message broker (MOM).
        iv) From the drop down list, select the type javax.jms.Queue
        v) Click OK.
    e) You can now leave of the admin console.

Build an application consisting of a web application and a Message Driven Bean

4) Return to Netbeans and choose Projects.
5) Select File then New Project
   a) Select Java EE and Enterprise Application and click Next
   b) The project name is MyJMSProject, and click Next.
   c) Create an EJB Module and a Web Application Module using the names provided.
   d) Click Finish.

   Notice that under the Projects tab you now have three new components. First is the overall project named MyJMSProject. You actually do not work with this directly. Rather you will develop within the individual components of MyJMSProject-ejb and MyJMSProject-war.

   MyJMSProject-ejb: This is the Enterprise Java Bean component where you will build the Queue listener called a Message Driven Bean. It will implement an onMessage method that will be called whenever there is a message in the queue it is listening to.

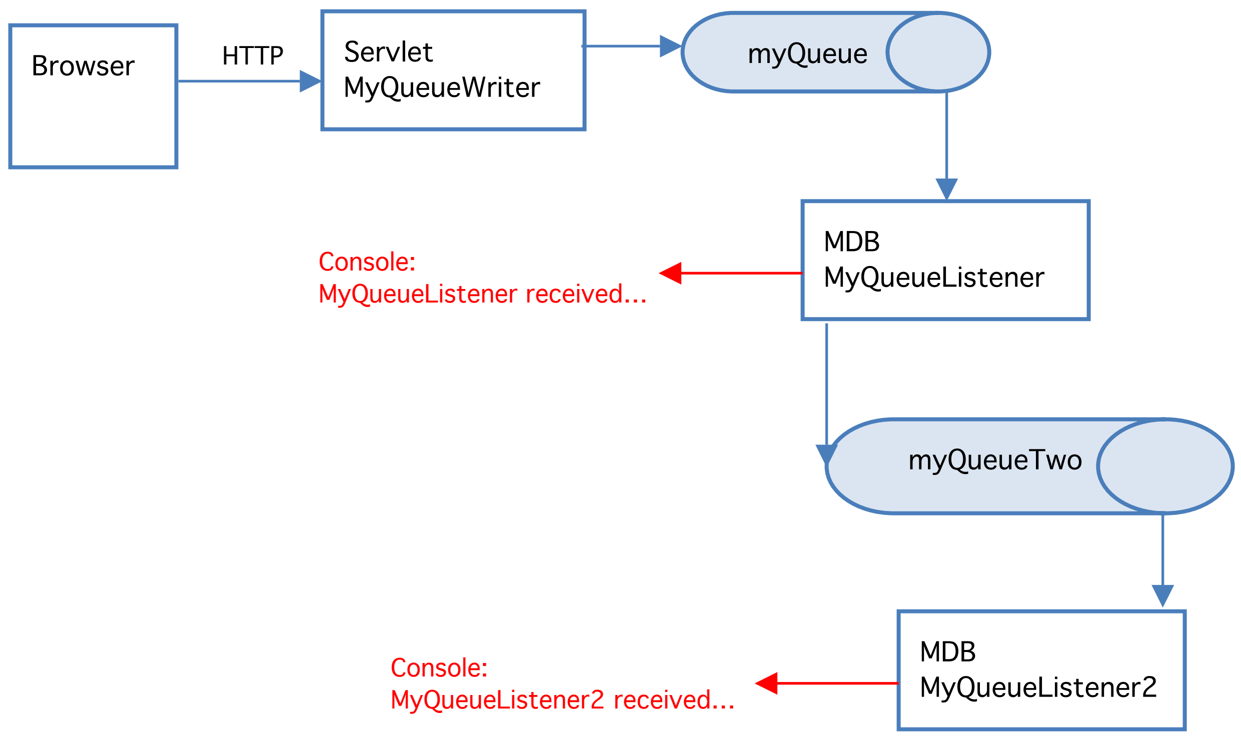
   MyJMSProject-war: This is the web application component. This is the same as the projects you have been developing this semester, and it is where the servlets and html/jsp code lives.

6) Populate the EJB component with a Message Driven Bean.
   a) From the Project View, right-click MyJMSProject-ejb.
   b) Select New Message Driven Bean.
      - EJB Name: MyQueueListener
      - Package name: edu.cmu.heinz.ds.queuelistener
      - Select the server destination as jms/myQueue.
        - Notice that this is the JNDI name of the queue that you created earlier.
   c) Select Next then Finish and you should see a default Message Driven Bean.
7) Replace the MyQueueListener onMessage method (**and only that method, not the whole file**) with the code in:
   http://www.andrew.cmu.edu/course/95-702/Labs/Lab9-JMS/MyQueueListener.java
   You will also have to add in the required imports:
   import javax.jms.TextMessage;
   import javax.jms.JMSException;
8) Now build a web application that sends messages to the queue.
   a) In the Project View, expand the MyJMSProject-war.
   b) Expand Web Pages
   c) Right click on index.html and delete it
   d) Right click on Web Pages and create a New JSP
      - Give it the file name index (don't name it index.jsp, the NetBeans wizard will add the jsp)
   e) Replace index.jsp with the code found in:
      http://www.andrew.cmu.edu/course/95-702/Labs/Lab9-JMS/index.jsp
      - Make sure the "@page" directive is on the first line of the file
9) Create a servlet to take text from the browser and send it to the Queue for the Message Driven Bean
   a) In the Project View, select MyJMSProject-war.
   b) Right click on it and choose New Servlet.
   c) The servlet name is MyQueueWriter.
   d) The package name is edu.cmu.heinz.ds.myqueuewriter.

e) Choose Next
f) ***UN-Select*** (if it is selected) "Add information to deployment descriptor (web.xml)" and then Finish.
g) Replace **all of the** MyQueueWriter.java with the code found in:
http://www.andrew.cmu.edu/course/95-702/Labs/Lab9-JMS/MyQueueWriter.java

10) From the Project View, right click MyJMSProject-ejb and select "deploy".
    - This deploys the Message Driven Bean to the EJB container
11) From the Project View, right click on MyJMSProject-war and select "run".
    - This both deploys the servlet and launches the browser visiting the index.jsp page.
12) When MyQueueListener gets the message from the Queue, it simply prints the content to the console. Therefore, check your GlassFish console and verify the message you typed was receive by MyQueueListener.
    - If you typed "Hello World" in the browser form, you should see the following line on the console:
    - INFO: MyQueueListener received: Hello world

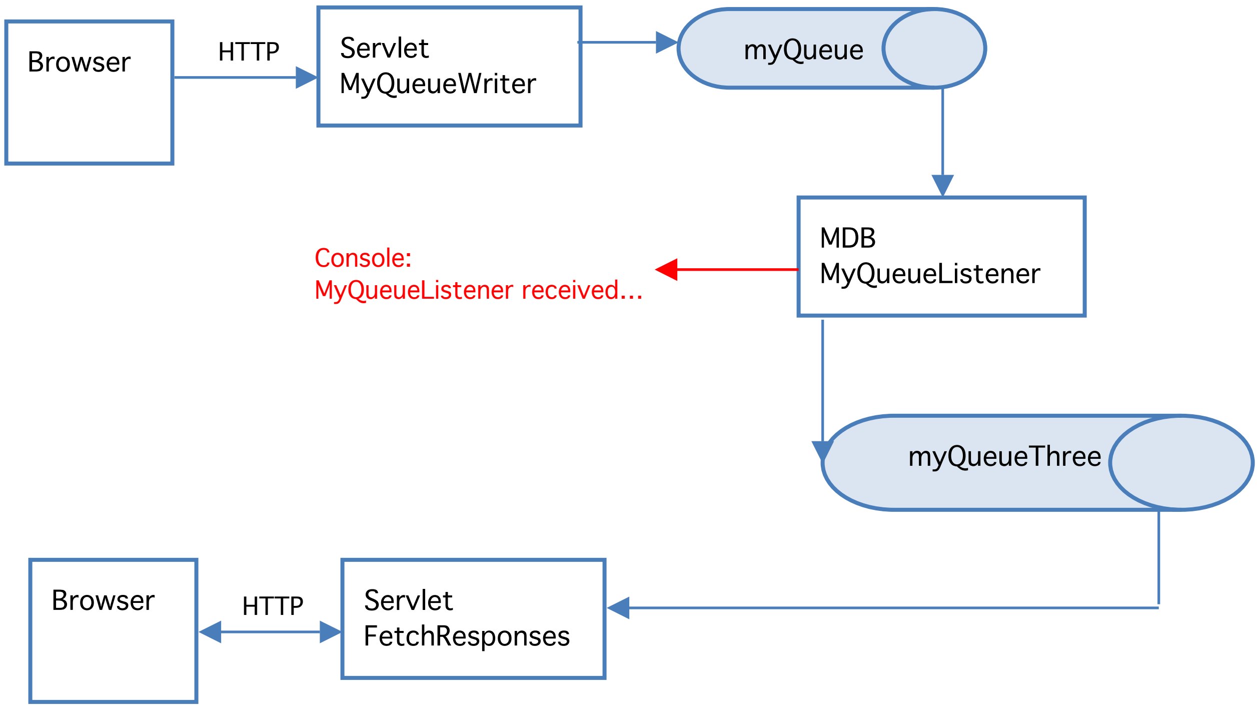
Task 2 – Two Phase Process



Modify your project such that:
1) MyQueueListener adds text to the message (e.g. <message text>+ " after processing by MyQueueListener"), and sends the new message to a new Queue named jms/myQueueTwo, with Physical Destination Name myQueueTwo
2) Create the new myQueueTwo in the GlassFish Admin Console

3) Create MyQueue2Listener modeled after the MyQueueListener, but listening to the new Queue.
4) Modify the string written to the console so you know the string is coming from the MyQueue2Listener.
5) Add code to MyQueueListener to send the message to myQueueTwo. The code to do this is similar to that used in MyQueueWriter to write to a myQueue.

Task 3 – Servlet reading from a Queue



1) Create a new Queue jms/myQueueThree
2) Modify MyQueueListener to write to jms/myQueueThree instead of myQueueTwo. Write a new servlet, FetchResponses that reads all available messages in jms/myQueueThree and displays them on a web page.
   a) If no messages are available, the servlet clearly state so on the response page.
   b) If there are one or more messages in the Queue, all should be displayed.
   - **Be sure to *start* the connection before trying to consume message from it.**
   - The code to read from a Queue is very similar to that of writing to it, only:
     - Use createConsumer instead of createProducer
     - Use receive to get a message instead of send
     - Use receive(1000) to have the receive time out if no messages are available.
     - Each receive will only get one message, so repeat until no more messages are in the Queue.
     - See sample MessageConsumer code on JMS slide 45.

Extra Challenge - Task 4 – Topics (not required)

1) Create a new Topic named jms/myTopicOne
   (just like you would a Queue, but choose javax.jmx.Topic instead).
2) Have MyQueueWriter write to the new Topic, instead of a Queue
3) Have ***both*** MyQueueListener and MyQueue2Listener subscribe to the Topic
4) Have MyQueueListener also write to jms/myQueueThree, the same way that MyQueue2Listener does.
5) Run the application, and notice the messages that MyQueueListener and MyQueue2Listener write to the console. Both should be receiving the message and both sending messages back via jms/myQueueThree to be picked up by FetchResponses.