Project Topics: Messaging and the Chandy-Lamport Snapshot Algorithm

This project has only one task: To add the Chandy-Lamport Snapshot Algorithm into a distributed system.

The distributed system has five players. Each player has a set of commodities which it trades with the other players. A player initiates a trade by making an offer to another randomly chosen player. The other player can accept or reject the trade. If they accept it, they pay with a commodity of their own. If they reject it, they send back the commodity. The trading action therefore is a fast series of offers, acceptances, rejections, and more offers.

Each of the 5 players is modeled as a Message Driven Bean. The code for each is nearly identical, except for its class name, the Queue it listens to, and an instance variable named myPlayerNumber. Each of the players instantiates a PITPlayerModel which does all the business (game) logic for the simulation.

All communication between the players is done by JMS Message Queues. Each player has its own Queue that it listens to. Other players can communicate with the player by sending a message to its Queue. (See the picture on left, or slide 38 of the Time and Global State lecture.)

A servlet and a Test Snapshot web page allows the system to be tested. Clicking on the Start Simulation button will start the simulation running. The servlet will send a series of messages to each Player’s Queue. First it sends a Reset.HALT message to each Player and awaits its acknowledgement response. This ensures that the players stop trading if they had been actively doing so. Next it sends a Reset.CLEAR message to each Player to have them reset their data structures and awaits their responses. Once all five Players have
been reset, it sends a NewHand message to each with a set of commodities. In this way, each Player is assigned its own initial set of commodities. These commodities are also known as *cards*. As soon as each Player receives its NewHand, it begins trading.

Trading continues until the maxTrades threshold is hit. This can be adjusted in the PITPlayerModel so the trading does not go on forever. The trading can also be stopped by clicking the Halt Simulation button on the Test Snapshot page.

A new round of trading can then be started by using the PITsnapshot servlet again. Initially, the Test Snapshot page will show a list of Snapshot Failed messages. This is normal because the snapshot has not yet been implemented. You will be implementing it.

### Setting up Queues

It is important that you set up the following JMS resources using the following names so that the system will work without extra work on your part, and so the TAs can run and test you solution on their laptops.

1. Create a JMS Connection Factory named: jms/myConnectionFactory
   (You should already have this resource from the previous lab. If so, you do not have to replicate it.)

2. Create the following JMS Destination Resources (Be careful with spelling!)

<table>
<thead>
<tr>
<th>JNDI Name</th>
<th>Physical Destination Name</th>
<th>Resource Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>jms/PITmonitor</td>
<td>PITmonitor</td>
<td>javax.jms.Queue</td>
</tr>
<tr>
<td>jms/PITsnapshot</td>
<td>PITsnapshot</td>
<td>javax.jms.Queue</td>
</tr>
<tr>
<td>jms/PITplayer0</td>
<td>PITplayer0</td>
<td>javax.jms.Queue</td>
</tr>
<tr>
<td>jms/PITplayer1</td>
<td>PITplayer1</td>
<td>javax.jms.Queue</td>
</tr>
<tr>
<td>jms/PITplayer2</td>
<td>PITplayer2</td>
<td>javax.jms.Queue</td>
</tr>
<tr>
<td>jms/PITplayer3</td>
<td>PITplayer3</td>
<td>javax.jms.Queue</td>
</tr>
<tr>
<td>jms/PITplayer4</td>
<td>PITplayer4</td>
<td>javax.jms.Queue</td>
</tr>
</tbody>
</table>
**CRITICAL:** Set GlassFish Web Container MDB Settings

The Glassfish Web Container will typically instantiate multiple Message Driven Beans when there are multiple messages in any Queue. Therefore, the web container could create multiple instantiations of each Player in our system. This can lead to undesirable race conditions. Therefore we need to ensure that only one MDB will be instantiated for each Player. This is done by setting the Maximum Pool Size to 1.

To do this, open the GlassFish Admin Console.

- Navigate on the left to open *Configurations*, and then *server-config*.
- Select *EJB Container*
- On the top of the right panel, find and choose *MDB Settings*. (Not EJB Settings!)
- Set:
  - Initial and Minimum Pool Size: 0
  - Maximum Pool Size: 1
  - Pool Resize Quantity: 1
- Click Save

**Installing the system**

Download Spring2017Project6.zip from the course schedule but DO NOT UNZIP IT. Use File-> Import Project -> From Zip to find and open the project within NetBeans.

*Resolve* any project problems

Clean and Build Spring2017Project6-ejb

Deploy Spring2017Project6-ejb

Clean and Build Spring2017Project6-war

Deploy Spring2017Project6-war

CHECK under the Services tab, expand GlassFish Server 4, expand Applications and confirm that the Spring2017Project6-ejb is deployed. Sometimes it needs to be deployed a second time. (I don't know why, and have never needed to deploy it a third time.)
Testing

Open a web browser and browse to the URL:
http://localhost:<port number>/Spring2017Project6-war/

Click on the button to start the simulation and after a short while you will see the response: "PIT has been initiated"

Go to the Server Log in Glassfish and review the output that is produced by the system. It should look something like this (only many more lines):

```
Info: Servlet sending Reset HALT to PITplayer0
Info: PITplayer0 received Reset HALT
Info: Servlet sending Reset HALT from PITplayer0 ACKNOWLEDGED
Info: Servlet sending Reset HALT to PITplayer1
Info: PITplayer1 received Reset HALT
Info: Servlet sending Reset HALT from PITplayer1 ACKNOWLEDGED
Info: Servlet sending Reset HALT to PITplayer2
Info: PITplayer2 received Reset HALT
Info: Servlet sending Reset HALT from PITplayer2 ACKNOWLEDGED
Info: Servlet sending Reset CLEAR to PITplayer0
Info: PITplayer0 received Reset CLEAR
Info: Servlet sending Reset CLEAR from PITplayer0 ACKNOWLEDGED
Info: Servlet sending Reset CLEAR to PITplayer1
Info: PITplayer1 received Reset CLEAR
Info: Servlet sending Reset CLEAR from PITplayer1 ACKNOWLEDGED
Info: Servlet sending Reset CLEAR to PITplayer2
Info: PITplayer2 received Reset CLEAR
Info: Servlet sending Reset CLEAR from PITplayer2 ACKNOWLEDGED
Info: Servlet sending newhand to 0
Info: PITplayer0 new hand: size: 13 wheat wheat wheat wheat wheat wheat wheat wheat wheat wheat wheat wheat wheat
Info: PITplayer0 tradeCount: 0
Info: PITplayer0 offered: wheat to player: 3
Info: Servlet sending newhand to 1
Info: PITplayer1 new hand: size: 13 corn corn corn corn corn corn corn corn corn corn corn corn corn corn
Info: PITplayer1 tradeCount: 0
Info: PITplayer1 offered: corn to player: 2
```
This is a global history of the actions being taken by the 5 players. It will eventually stop when each Player hits 20000 trades or you click Halt Simulation.

Near the end of the global history will be lines similar to:

INFO: Servlet Initiating Snapshot
INFO: PITplayer3 received unknown Message type
INFO: Servlet: Not all players reported, giving up after 0

The first message is from the Servlet indicating that it is about to send a marker message into the queue of one of the PITplayers. PITplayer3 then reports that it got...
a message of unknown type (because it is of type Marker and it doesn’t know how to handle them (yet)). The final line is from the Servlet again reporting that it has not received snapshot messages back from all of the players. At this point these console messages make sense because you have not implemented the snapshot algorithm yet.

Back in the browser, test results from 10 snapshots will be added to the window. It will look like the screenshot on right.

Again, the snapshots are failing because the snapshot code has not yet been implemented. That is your task; implement the snapshot code.

This web page is reusable without re-loading. (It uses AJAX.) So at any time you can just click on Start Snapshot to start the next snapshot.

Do not use Internet Explorer to test with the Test Snapshot page. IE erroneously caches the AJAX requests and you will get invalid results. Use Chrome or Safari instead.

If you get simulation to hit 20000 trades, you have completed the Commodity Trading Simulation lab. Show a TA for credit, and you are now ready to start the project.

Task – Implement the Chandy Lamport Snapshot Algorithm

In class we discussed the Chandy Lamport Snapshot Algorithm. Implement this algorithm in the system so that you can check if any commodities have been added-to or lost-from the system. Since there are 13 of each commodity given out by PITsnapshot, and none or consumed or added, there should always be a steady state of 13 of each commodity shared between the 5 Players.

Some pieces have been provided to you for this task:

The Marker class is defined for passing as the Marker in the snapshot algorithm. The servlet PITsnapshot will initiate the snapshot by sending a Marker to some player. (All 5 Players run the same PITPlayeModel code, so the PITsnapshot should be able to initiate the snapshot by sending to any of the 5 Players.)

PITsnapshot will then wait and read from the PITsnapshot Queue. Each Player should send a message back to PITsnapshot via that Queue. The content of that message should be an ObjectMessage, and the Object should be a HashMap of commodities and
counts (see the code for details). Add to the HashMap the identify of who the snapshot is coming from in the format: state.put("Player", myPlayerNumber);

Finally, the PITsnapshot servlet will report the sums of each commodity back to the browser.

The picture to the right shows only the first two snapshots. In total 10 will be attempted.

The snapshot is successful if the number of each commodity is 13.
Until your code is correct, you will probably see cases where there is undercounting (commodities < 13) and overcounting ( > 13). Your snapshot code should repeatedly pass all 10 tests.

Therefore the core of this task is to modify ONLY the code in PITPlayerModel.java. (No other file should be edited.) Modify the player model so that it implements the snapshot algorithm for the PITplayers and pass the results to the PITsnapshot servlet.

Work independently

You should work independently on this project. You might have heard that in past semesters students could work in pairs. Unfortunately the overwhelming assessment we got was that this did not lead to good peer-programming practices, but rather one person did the work and both took credit.

You may use your remaining grace days on this project. The project is due at 5pm, but grace days will extend until 11:59pm.
What to turn in

1. Create a directory named with your Andrew ID (and only your Andrew id).

2. Take screen shots of a successful snapshot (because of its length, it will probably take more than one) and put it into this new directory.

3. Copy PITPlayerModel.java (only!) into the directory. This should have been the only file you modified.
   - You should not include your whole project, only PITPlayerModel.java.

4. Zip the directory containing the screen shots and the PITPlayerModel.java

   Therefore your zip file should contain ONLY:
   - A few screenshots
   - PITPlayerModel.java

5. Submit the zipped file to Blackboard.

   (Note: You must have used the correctly named Connection Factory and Queues to get full credit.)

Celebrate!

After you turn in your Project 6, be sure to join the Heinz College End of Year Cinco de Mayo Picnic! It is from 5pm – 8pm on Friday May 5 in the Schenley Park Vietnam Veteran’s Pavilion.

There will be hot dogs, hamburgers, veggie dogs, an ice cream bar and drinks!