95-702 Distributed Systems
Project 6
Due: Friday, December 9, 11:59 PM
** Pair Programming Permitted **
(See below for details)

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. Trades are more like gifts than exchanges. When each player receives a commodity from someone, it gives one of its commodities to another player. It might be the same player, or it may be another player. It picks who to give the Trade to randomly. The trading action therefore is a fast series of accepting commodities from the others and giving commodities to others.

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.

A servlet allows the system to be initialized (PITsnapshot.java). This servlet will send a series of two messages to each Player's Queue. First it sends a Reset message to each Player and awaits its acknowledgement response. Once all five Players have been reset, it sends a NewHand message to each of the Players 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.

A new round of trading can then be started by using the PITsnapshot servlet again.
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 your 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

<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 Fall2016Project6.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 Fall2016Project6-ejb
Deploy Fall2016Project6-ejb
Clean and Build Fall2016Project6-war
Deploy Fall2016Project6-war

CHECK under the Services tab, expand GlassFish Server 4, expand Applications and confirm that the Fall2016Project6-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>/Fall2016Project6-war/

Test Snapshot

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):

```plaintext
Info: 4:16:12 PM Reset HALT of PITplayer0
Info: PITplayer0 received Reset HALT
Info: 4:16:12 PM Reset HALT of PITplayer0 ACKNOWLEDGED
Info: PITplayer1 received Reset HALT
Info: 4:16:12 PM Reset HALT of PITplayer1 ACKNOWLEDGED
Info: PITplayer2 received Reset HALT
Info: 4:16:12 PM Reset HALT of PITplayer2 ACKNOWLEDGED
Info: PITplayer3 received Reset HALT
Info: 4:16:12 PM Reset HALT of PITplayer3 ACKNOWLEDGED
Info: PITplayer4 received Reset HALT
Info: 4:16:12 PM Reset HALT of PITplayer4 ACKNOWLEDGED
Info: 4:16:12 PM Reset CLEAR of PITplayer0
Info: PITplayer0 received Reset CLEAR
Info: 4:16:12 PM Reset CLEAR of PITplayer0 ACKNOWLEDGED
Info: PITplayer1 received Reset CLEAR
Info: 4:16:12 PM Reset CLEAR of PITplayer1 ACKNOWLEDGED
Info: PITplayer2 received Reset CLEAR
Info: 4:16:12 PM Reset CLEAR of PITplayer2 ACKNOWLEDGED
Info: PITplayer3 received Reset CLEAR
Info: 4:16:12 PM Reset CLEAR of PITplayer3 ACKNOWLEDGED
Info: PITplayer4 received Reset CLEAR
Info: 4:16:12 PM Reset CLEAR of PITplayer4 ACKNOWLEDGED
Info: 4:16:12 PM: sending newhand to 0
Info: PITplayer0 new hand: size: 13 wheat wheat wheat wheat wheat wheat wheat wheat wheat wheat wheat wheat
Info: PITplayer0 tradeCount: 0
Info: PITplayer0 sending: wheat to player: 3
Info: 4:16:12 PM: sending newhand to 1
Info: PITplayer1 new hand: size: 13 corn corn corn corn corn corn corn corn corn corn corn corn corn
Info: PITplayer1 tradeCount: 0
Info: PITplayer1 sending: corn to player: 2
Info: 4:16:12 PM: sending newhand to 2
Info: PITplayer3 received: wheat from player: 0
Info: PITplayer3 hand: size: 1 wheat
Info: PITplayer3 tradeCount: 0
Info: PITplayer2 new hand: size: 13 coffee coffee coffee coffee coffee coffee coffee coffee coffee coffee coffee coffee
Info: PITplayer2 tradeCount: 0
Info: PITplayer2 sending: coffee to player: 4
Info: 4:16:12 PM: sending newhand to 3
Info: PITplayer3 new hand: size: 14 wheat soybeans soybeans soybeans soybeans soybeans soybeans soybeans soybeans soybeans
Info: PITplayer3 sending: wheat to player: 1
Info: PITplayer4 received: coffee from player: 2
Info: PITplayer4 tradeCount: 0
Info: PITplayer4 sending: coffee to player: 0
Info: 4:16:12 PM: sending newhand to 4
Info: PITplayer2 new hand: size: 13 coffee oats oats oats oats oats oats oats oats oats oats oats oats oats
Info: PITplayer2 tradeCount: 0
Info: PITplayer2 sending: coffee to player: 1
Info: 4:16:12 PM: sending newhand to 1
Info: PITplayer1 received: wheat from player: 3
Info: PITplayer1 hand: size: 13 corn corn corn corn corn corn corn corn corn corn corn corn corn
Info: PITplayer1 sending: corn to player: 3
Info: PITplayer0 received: coffee from player: 4
Info: PITplayer0 tradeCount: 0
Info: PITplayer0 sending: coffee to player: 2
Info: PITplayer3 received: coffee from player: 1
Info: PITplayer3 tradeCount: 0
Info: PITplayer3 sending: soybeans to player: 4
Info: PITplayer4 received: coffee from player: 2
Info: PITplayer4 tradeCount: 0
Info: PITplayer4 sending: oats to player: 3
Info: PITplayer2 received: wheat from player: 0
```

This is a global history of the actions being taken by the 5 players. It will eventually stop when each Player hits 20000 trades.

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

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

The first message is from the PITsnapshot 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.

If you get simulation to hit 20000 trades, you have completed the lab (and are 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, 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.

Peer Programming

For this project, you can work in teams of two. If you prefer, you can work alone. You can fully collaborate and turn in only one solution. You cannot discuss your project with others beyond your teammate. If working as a team, you should only turn in the project to Blackboard under the ID of the teammate whose Andrew ID comes first sorted alphabetically.

If you are working in pairs, it is a bad strategy to try to divide and conquer. There is only one problem to solve, not two to be done in parallel. Therefore use pair programming; both of you sit together, on the same computer, working on the
problem together. There are a lot of benefits to this type of development in terms of lower errors, higher satisfaction, and increased learning.

Any student who has not used the one-late-assignment-allowance can use it on this project. If both students have not used their allowance already, then neither will be penalized if they turn the assignment in up to a week late. However, if a student has already used the allowance, they will be penalized 10% per day late. So if you have already used your allowance, you should probably not pair up with someone who still has theirs to use!

What to turn in

1. Create a directory called andrew1-andrew2 where andrew1 and andrew2 are the teammates andrew ids. If Mike and Joe were working together, this would be joemertz-mm6. If Joe was working alone it would be joemertz.

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.

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.
   - If you worked as a team, ONLY ONE STUDENT SHOULD SUBMIT

6. Complete the Peer Review Assignment on Blackboard
   - It will have three questions:
     o What was your percent effort
     o What was your partner’s name
     o What was your partner’s percent effort.

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