95-702 Distributed Systems

Project 6

Assigned: Monday, November 26

Due: Saturday, December 8, 11:59:59 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 three 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 3 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.

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 (PITinit.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 three 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 all three Players so that the trading does not go on forever.

A new set of trading can then be started by using the PITinit 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 you solution on their laptops.

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

JNDI Name	Physical Destination Name	Resource Type
jms/PITmonitor	PITmonitor	javax.jms.Queue
jms/PITsnapshot	PITsnapshot	javax.jms.Queue
jms/PITplayer0	PITplayer0	javax.jms.Queue
jms/PITplayer1	PITplayer1	javax.jms.Queue
jms/PITplayer2	PITplayer2	javax.jms.Queue

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.*
- Set:
  - Initial and Minimum Pool Size: 0
  - Maximum Pool Size: 1
  - Pool Resize Quantity: 1
- Click Save

Installing the system

Download Fall2012Project6.zip from the course schedule and upzip it in your NetBeans project directory.

Use File-> Open Project to find and open the project within NetBeans.

Expand the Fall2012Project6 project

Expand the Java EE Modules

Double-click on Fall2012Project6-war.war to load it into NetBeans

Double-click on Fall2012Project6-ejb.jar to load it into NetBeans

Clean and Build Fall2012Project6-ejb

Deploy Fall2012Project6-ejb

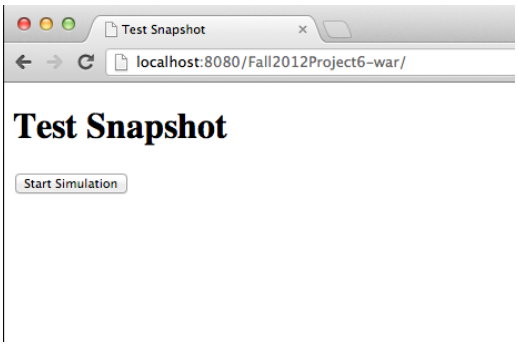
Clean and Build Fall2012Project6-war

Deploy Fall2012Project6-war

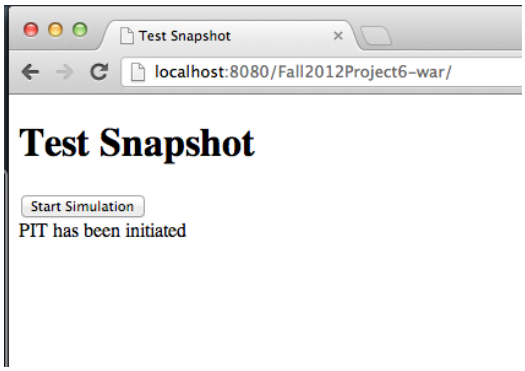
Testing

Open a web browser and browse to the URL:

http://localhost:<port number>/Fall2012Project6-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:

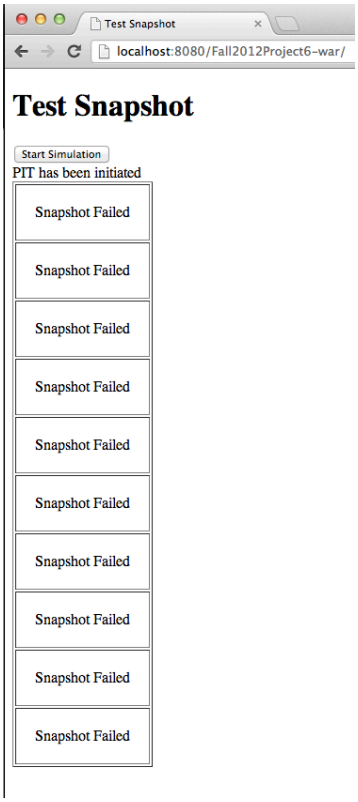
```
INFO: 1:06:35 PMReset of PITplayer0
INFO: 1:06:35 PMReset of PITplayer0 ACKNOWLEDGED
INFO: 1:06:35 PMReset of PITplayer1
INFO: PITplayer0 received reset
INFO: PITplayer1 received reset
INFO: 1:06:35 PMReset of PITplayer1 ACKNOWLEDGED
INFO: 1:06:35 PMReset of PITplayer2
INFO: PITplayer2 received reset
INFO: 1:06:35 PMReset of PITplayer2 ACKNOWLEDGED
INFO: 1:06:35 PM: sending newhand to 0
INFO: PITplayer0 new hand: size: 9 rice rice rice rice rice rice rice rice rice
INFO: PITplayer0 tradeCount: 0
INFO: 1:06:35 PM: sending newhand to 1
INFO: PITplayer0 sending: rice to player: 1
INFO: PITplayer1 new hand: size: 9 oil oil oil oil oil oil oil oil oil
INFO: PITplayer1 tradeCount: 0
INFO: PITplayer1 sending: oil to player: 2
INFO: 1:06:35 PM: sending newhand to 2
INFO: PITplayer1 received: rice from player: 0
INFO: PITplayer1 hand: size: 9 oil oil oil oil oil oil oil oil rice
INFO: PITplayer1 sending: oil to player: 2
INFO: PITplayer2 new hand: size: 9 gold gold gold gold gold gold gold gold gold
INFO: PITplayer2 tradeCount: 0
INFO: PITplayer2 sending: gold to player: 1
INFO: PITplayer2 received: oil from player: 1
INFO: PITplayer2 hand: size: 9 gold gold gold gold gold gold gold gold oil
INFO: PITplayer2 sending: gold to player: 1
INFO: PITplayer1 received: gold from player: 2
INFO: PITplayer1 hand: size: 9 oil oil oil oil oil oil oil rice gold
INFO: PITplayer1 sending: oil to player: 2
```

This is a global history of the actions being taken by the 3 players. It will eventually stop when each Player hits 1000 trades.

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

At this point the snapshots are failing because the snapshot code has not yet been implemented. That is your task.

This 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.



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 9 of each commodity given out by PITinit, there should always be a steady state of 9 of each commodity shared between the 3 Players.

Some pieces have been provided to you for this task:

The Market class is defined for passing as the Marker in the snapshot algorithm.

The servlet PITsnapshot will initiate the snapshot by sending a Marker to PITplayer0. (Remember that all 3 Players should have the same code, so the PITsnapshot should be able to initiate the snapshot by sending to any of the 3 Players.)

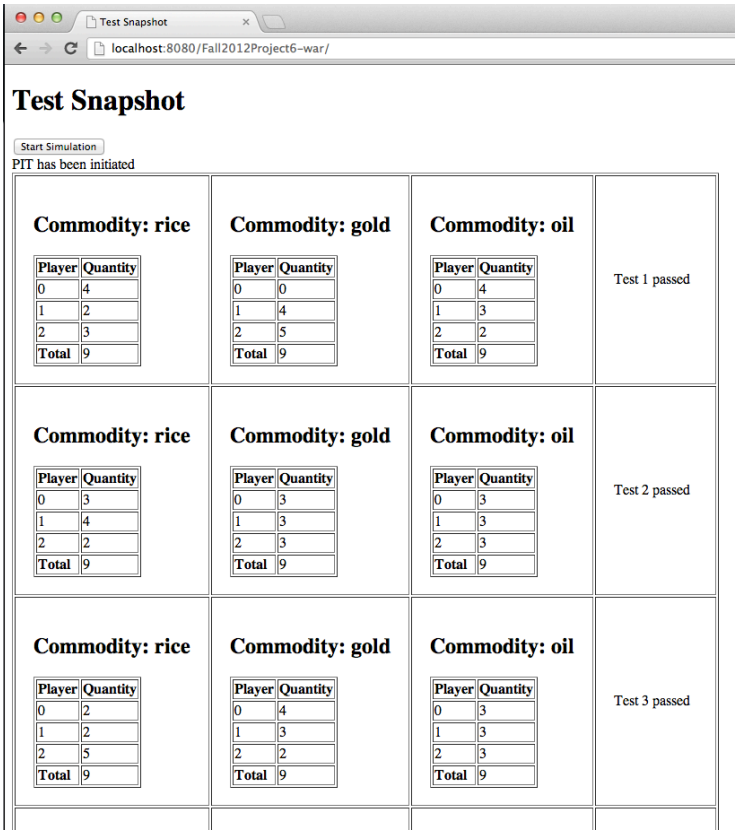
PITsnapshot will then wait and read from the PITsnapshot Queue. Each Player should send a message back to PITsnapshot via that Queue. The content of that message should be an ObjectMessage, and the Object should be a HashMap of commodities and counts (see the code for details).

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

The snapshot at right show only the first 3 snapshots. In total 10 will be attempted.

The snapshot is successful if the number of each commodity is 9. Until your code is correct, you will probably see cases where there is undercounting (commodities < 9) and overcounting (> 9). Your snapshot code should repeatedly pass all 10 tests.

NOTE: The "Player" number reported here is just the order of how responses were received from the 3 players. It is **not** the actual player number.



Therefore the core of this task is to modify the code in PITplayer0.java, PITplayer1.java, and PITplayer2.java. (No other files should need to be changed.)

Modify these Players so that they implement the snapshot algorithm 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.

What to turn in

1. Take a single screen shot of a successful snapshot (PITsnapshot) and add it to your NetBeans project directory
2. Zip your NetBeans project directory
3. Submit the zipped file to Blackboard.
    - If you worked as a team, ONLY ONE STUDENT SHOULD SUBMIT
4. Complete the Peer Review Assignment on Blackboard
    - It will have three questions:
        - What was your percent effort
        - What was your partner's name
        - What was your partner's percent effort.

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

Blackboard Forum

Please post your questions to the Blackboard Forum.

Please check the Blackboard Forum for suggestions.