95-702 Distributed Systems
Project 6

Due: Friday, May 3, 5:00 PM

NOTE: This is **not** due at midnight, but 5pm!

Project Topics: Messaging and the Chandy-Lamport Snapshot Algorithm

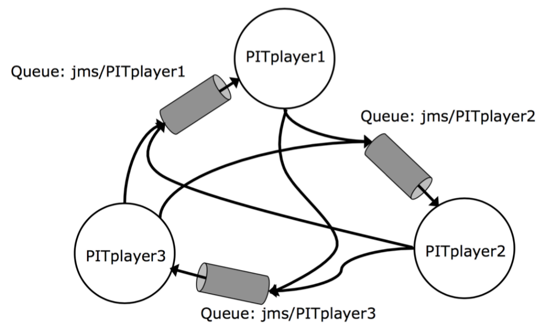
This project has three tasks:

- Add monopoly-seeking behavior to the players in a simulation.
- Add the Chandy-Lamport Snapshot Algorithm into a distributed system.
- Use the Chandy-Lamport Snapshot to understand the behavior of a system.

The distributed system is loosely based on the card game *Pit.*
Wikipedia: https://en.wikipedia.org/wiki/Pit_(game)
Amazon: https://www.amazon.com/NEW-Pit-Card-Game-Winning/dp/B0015H1A3S

Our simulation of the game has five players. Each player has a set of commodities that 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.



In the trading simulation, the channels are implemented as JMS Queues. This meets the Chandy & Lamport snapshot algorithm assumption that there are channels from each player to every other player, and that the channels are First-In-First-Out.
All players are implemented as Message Drive Beans (aka Queue Listeners)

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.)

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 your solution on their laptops.

1. Create a JMS Connection Factory named: jms/myConnectionFactory

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

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
jms/PITplayer3	PITplayer3	javax.jms.Queue
jms/PITplayer4	PITplayer4	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.* (Not EJB Settings!)
- Set:
  - Initial and Minimum Pool Size: 0
  - Maximum Pool Size: 1
  - Pool Resize Quantity: 1
- Click Save

Installing the system

Download Spring2019Project6.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 Spring2019Project6-ejb

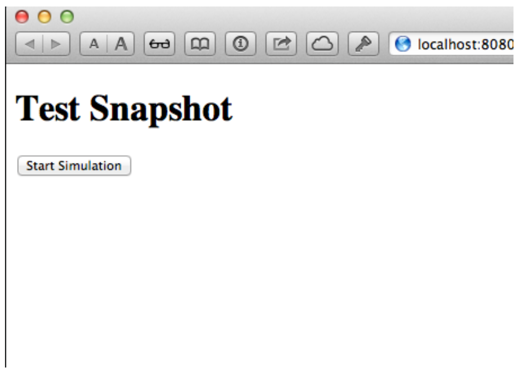
Deploy Spring2019Project6-ejb

Clean and Build Spring2019Project6-war

Deploy Spring2019Project6-war

CHECK under the Services tab, expand GlassFish Server, expand Applications and confirm that the Spring2019Project6-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

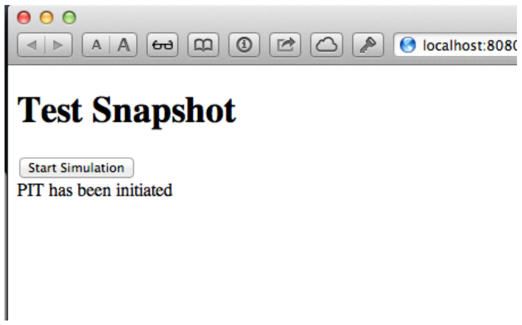


Test Snapshot

Open a web browser and browse to the URL:

http://localhost:<port number>/Spring2019Project6-war/

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



Test Snapshot

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 and different commodities):

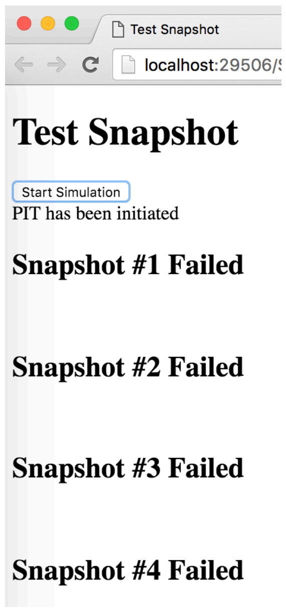
```
Info:   Servlet sending Reset HALT to PITplayer0
Info:   PITplayer0 received Reset HALT
Info:   Servlet Reset HALT from PITplayer0 ACKNOWLEDGED
Info:   Servlet sending Reset HALT to PITplayer1
Info:   PITplayer1 received Reset HALT
Info:   Servlet Reset HALT from PITplayer1 ACKNOWLEDGED
Info:   Servlet sending Reset HALT to PITplayer2
Info:   PITplayer2 received Reset HALT
Info:   Servlet Reset HALT from PITplayer2 ACKNOWLEDGED
Info:   Servlet sending Reset HALT to PITplayer3
Info:   PITplayer3 received Reset HALT
Info:   Servlet Reset HALT from PITplayer3 ACKNOWLEDGED
Info:   Servlet sending Reset HALT to PITplayer4
Info:   PITplayer4 received Reset HALT
Info:   Servlet Reset HALT from PITplayer4 ACKNOWLEDGED
Info:   Servlet sending Reset CLEAR to PITplayer0
Info:   PITplayer0 received Reset RESET
Info:   Servlet Reset CLEAR from PITplayer0 ACKNOWLEDGED
Info:   Servlet sending Reset CLEAR to PITplayer1
Info:   PITplayer1 received Reset RESET
Info:   Servlet Reset CLEAR from PITplayer1 ACKNOWLEDGED
Info:   Servlet sending Reset CLEAR to PITplayer2
Info:   PITplayer2 received Reset RESET
Info:   Servlet Reset CLEAR from PITplayer2 ACKNOWLEDGED
Info:   Servlet sending Reset CLEAR to PITplayer3
Info:   PITplayer3 received Reset RESET
Info:   Servlet Reset CLEAR from PITplayer3 ACKNOWLEDGED
Info:   Servlet sending Reset CLEAR to PITplayer4
Info:   PITplayer4 received Reset RESET
Info:   Servlet Reset CLEAR from PITplayer4 ACKNOWLEDGED
Info:   Servlet sending newhand to 0
Info:   PITplayer0 new hand: size: 15 Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt
Info:   PITplayer0 tradeCount: 0
Info:   PITplayer0 offered: Cobalt to player: 1
Info:   Servlet sending newhand to 1
Info:   PITplayer1 new hand: size: 15 Copper Copper Copper Copper Copper Copper Copper Copper Copper Copper Copper Copper
```

```
Info:   PITplayer1 tradeCount: 0
Info:   PITplayer1 offered: Copper to player: 3
Info:   Servlet sending newhand to 2
Info:   PITplayer2 new hand: size: 15 Nickel Nickel Nickel Nickel Nickel Nickel Nickel Nickel Nickel Nickel Nickel Nickel
Info:   PITplayer2 tradeCount: 0
Info:   PITplayer2 offered: Nickel to player: 3
Info:   Servlet sending newhand to 3
Info:   PITplayer1 received offer of: Cobalt from player: 0
Info:   PITplayer1 accepting offer and paying with: Copper to player: 0
Info:   PITplayer1 hand: size: 11 Copper Copper Copper Copper Copper Copper Copper Copper Copper Copper Cobalt
Info:   PITplayer3 new hand: size: 15 Tin Tin Tin Tin Tin Tin Tin Tin Tin Tin Tin Tin
Info:   PITplayer3 tradeCount: 0
Info:   PITplayer3 offered: Tin to player: 0
Info:   Servlet sending newhand to 4
Info:   PITplayer4 new hand: size: 15 Zinc Zinc Zinc Zinc Zinc Zinc Zinc Zinc Zinc Zinc Zinc Zinc
Info:   PITplayer4 tradeCount: 0
Info:   PITplayer4 offered: Zinc to player: 2
Info:   PITplayer3 received offer of: Copper from player: 1
Info:   PITplayer3 accepting offer and paying with: Tin to player: 1
Info:   PITplayer3 hand: size: 11 Tin Tin Tin Tin Tin Tin Tin Tin Tin Tin Copper
Info:   PITplayer0 received offer of: Tin from player: 3
Info:   PITplayer0 accepting offer and paying with: Cobalt to player: 3
Info:   PITplayer0 hand: size: 11 Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt Cobalt Tin
Info:   PITplayer1 received: Tin as payment from player: 3
Info:   PITplayer1 hand: size: 15 Copper Copper Copper Copper Copper Copper Copper Copper Copper Copper Cobalt Tin
Info:   PITplayer1 offered: Copper to player: 3
Info:   PITplayer2 received offer of: Zinc from player: 4
Info:   PITplayer2 accepting offer and paying with: Nickel to player: 4
Info:   PITplayer2 hand: size: 11 Nickel Nickel Nickel Nickel Nickel Nickel Nickel Nickel Nickel Nickel Zinc
Info:   PITplayer3 received offer of: Nickel from player: 2
Info:   PITplayer3 accepting offer and paying with: Tin to player: 2
Info:   PITplayer3 hand: size: 11 Tin Tin Tin Tin Tin Tin Tin Tin Tin Copper Nickel
Info:   PITplayer0 received: Copper as payment from player: 1
```

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:   PITplayer2 received unknown Message type
INFO:   Servlet: Not all players reported, giving up after 0
```



Test Snapshot

PIT has been initiated

Snapshot #1 Failed

Snapshot #2 Failed

Snapshot #3 Failed

Snapshot #4 Failed

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. PITplayer2 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 – Add monopoly-seeking behavior to the players in a simulation

The goal of the original *Pit* card game is to corner a commodity market by having all 15 cards of that single commodity; for example, all 15 cards of Cobolt.

The simulation currently randomly originates trades. Add logic to the simulation so that each player will ***attempt*** to collect a monopoly of one commodity.

In order to do so:

- You can **only** modify PITPlayerModel.java.
- The logic must be generalizable and work for:
    - Any number of players
    - Any number of commodities
    - Any commodity names
    - Any number of commodity cards per player
- Your logic can refer to the information in a TenderOffer, including TenderOffer.sourcePlayer, to decide how to respond to an offer.
- Your logic can decide what playerNumber to tender a new offer to (currently new offers are tendered to a random other player). This playerNumber cannot be hard coded, however (e.g. always offer trades to hardcoded player 4).

Task – Implement the Chandy Lamport Snapshot Algorithm

In class, we discuss the Chandy Lamport Snapshot Algorithm. Implement this algorithm in the system so that you can check the state of players and commodities, such as what player is holding what commodities. Since there are 15 of each commodity given out by PITsnapshot, and none or consumed or added, there should always be a steady state of 15 of each commodity shared between the 5 Players. (Each Player, however, may have more or less than 15 commodities at any given time.)

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.)

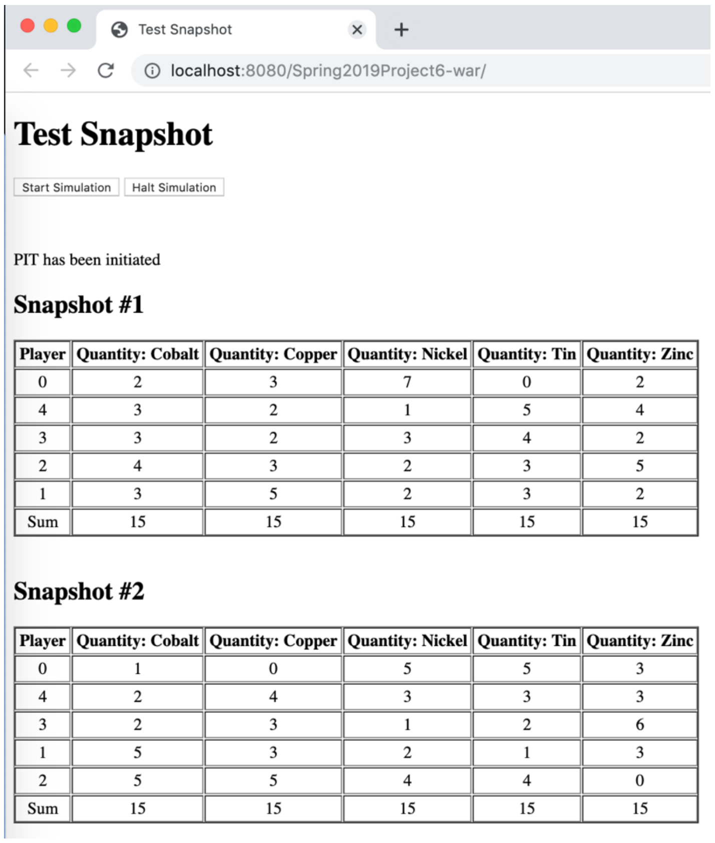
PITsnapshot will then wait and read from the PITsnapshot Queue. Each Player should send a message back to PITsnapshot via that Queue. The content of that message should be an ObjectMessage, and the Object should be a HashMap of commodities and counts (see the code for details). Add to the HashMap the 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 15. Until your code is correct, you will probably see cases where there is undercounting (commodities < 15) and overcounting (> 15). 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.



Test Snapshot

PIT has been initiated

Snapshot #1

Player	Quantity: Cobalt	Quantity: Copper	Quantity: Nickel	Quantity: Tin	Quantity: Zinc
0	2	3	7	0	2
4	3	2	1	5	4
3	3	2	3	4	2
2	4	3	2	3	5
1	3	5	2	3	2
Sum	15	15	15	15	15

Snapshot #2

Player	Quantity: Cobalt	Quantity: Copper	Quantity: Nickel	Quantity: Tin	Quantity: Zinc
0	1	0	5	5	3
4	2	4	3	3	3
3	2	3	1	2	6
1	5	3	2	1	3
2	5	5	4	4	0
Sum	15	15	15	15	15

Task – Use snapshots to understand the behavior of a system

Create a pdf document named MonopolyStrategyAnalysis.pdf with the following content:

A. Name and Andrew ID
B. Describe the strategy you implemented for having each player work toward a monopoly. Write this in prose so that a non-technical person would understand the approach you took for trading toward a monopoly. (200 words maximum)
C. Looking at your sequences of 10 snapshots, perhaps run several or many times, describe what you find interesting about any patterns of players moving toward achieving monopolies. How well did these patterns match or not match what you expected of your strategy? How well did your strategy result in one more players achieving monopolies? (400 words maximum)

Grace days

You may use your remaining grace days on this project, even though these days may go into finals week.

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. Put the file MonopolyStrategyAnalysis.pdf in the directory.
5. Zip the directory containing these files
   Therefore your zip file, named YourAndrewID.zip, should contain ONLY:
   - A few screenshots
   - PITPlayerModel.java
   - MonopolyStrategyAnalysis.pdf
6. Submit the zipped file to Blackboard.

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