To start a node
For each node, open a new terminal and type the following command:
```
java Main ../test/[Test File] [Peer ID]
```

To configure a test file
Each test file is formatted as follows:
```
[Peer ID] [Peer Type] [Peer IP]:[Peer Port] [Neighbor ID] [Neighbor IP]:[Neighbor Port]
```

For example, in OnlyBuyer.txt,
```
1 0 128.119.243.164:9100 2 128.119.243.168:9200
2 0 128.119.243.168:9200 1 128.119.243.164:9100
```

There are two nodes in the network with Peer ID 1 and 2. Both are Peer Type 0, or Buyer. They run on different edlab machines which are 128.119.243.164 (elnux2) on port 9100 and 128.119.243.168 (elnux3) on port 9200. Peer 1 and Peer 2 are each other's neighbor. One node can have multiple neighbors by separating Neighbor IDs and Neighbor IPs with comma respectively:

```
1 0 128.119.243.164:9100 3 128.119.243.175:9300
2 0 128.119.243.168:9200 3 128.119.243.175:9300
3 1 128.119.243.175:9200 1,2 128.119.243.164:9100,128.119.243.168:9200
```

Edlab IP

elnux2	128.119.243.164
elnux3	128.119.243.168
elnux7	128.119.243.175

Peer Type
0: Buyer, 1: Seller, 2: BuyerAndSeller

Interpret the logs
We use the following network as an example:
```
1 0 128.119.243.164:9100 2 128.119.243.168:9200
2 1 128.119.243.168:9200 1 128.119.243.164:9100
```
We use message enclosed with operation code so the receiver can deal with the

message accordingly. There are three types of operation: LOOKUP, REPLY, BUY. When a buyer is up, it is randomly assigned a product type and message id. We log out the information of the peer.

```
PeerID 1
NodeType = 0
myIP = 128.119.243.164:9100
neighborPeerID:
[2]
peerIPMap:
{2=128.119.243.168:9200}
Peer 1 initiated, type 0
```

Then the buyer forms a LOOKUP message and floods to all neighbors.

```
Buyer 1 LookUp product 1 MessageID 95657
```

If the peer cannot find its neighbor, you should see the following message:

```
Connection refused to host: 128.119.243.168; nested exception is:
	java.net.ConnectException: Connection refused (Connection refused)
```

If the seller is found, the seller sends a message backward with REPLY:

```
Peer 2 Reply LookUp from peer 1 for product 1 MessageID 95657
```

If not, we log

```
No Seller found!
```

Then the buyer will build the connection directly with the seller by sending BUY message:

```
Peer 2 received BUY from peer 1 for product 1 and Sold it MessageID 95657
```

Test Cases

Not working cases

OnlyBuyer.txt (elnux2, elnux3)

```
No Seller found!
```

As there is no seller in the market, buyers can never find a seller.

OnlyBuyerAndSeller.txt (elnux2, elnux3)

```
No Seller found!
```

Since a BuyerAndSeller cannot buy and sell the same product, the network will eventually run into a deadlock if all nodes are selling the same product.

NodeWithNoNeighbor.txt (elnux2, elnux3)

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
	at Main.main(Main.java:29)
```

Every node should have a neighbor when building the network.

SelfNeighbored.txt (elnux2)

```
No Seller found!
```

There is only one buyer in the market so the deal will never be fulfilled.

Working cases

MBuyerConcurr.txt (elnux2, elnux3, elnux7)

```
Peer 3 Reply LookUp from peer 1 for product 1 MessageID 95657
Peer 3 received BUY from peer 1 for product 1 and Sold it MessageID 95657
```

We run two buyers on elnux2 and elnux3, and a seller on elnux7 to simulate concurrent request. Peer 1 or Peer 2 will receive a REPLY message from Peer 3. Then Peer 3 will receive a BUY message from Peer 1 or Peer 2. The product will be hold for one of the buyers until the seller receives the BUY message.

MSellerConcurr.txt (elnux2, elnux3, elnux7)

```
Peer 1 Reply LookUp from peer 3 for product 1 MessageID 95657
Peer 1 received BUY from peer 3 for product 1 and Sold it MessageID 95657
```

We run two sellers on elnux2 and elnux3, and a buyer on elnux7 to simulate concurrent request. Two sellers will receive the same message. If they both REPLY to the same buyer, only the first REPLY message will be handled. So the buyer will build connection to only one of the two sellers.

MSellerMBuyerConcurr.txt (elnux2, elnux3)

```
Peer 2 Reply LookUp from peer 1 for product 1 MessageID 95657
Peer 4 received BUY from peer 1 for product 1 and Sold it MessageID 95657
Peer 2 Reply LookUp from peer 3 for product 1 MessageID 75657
Peer 4 received BUY from peer 3 for product 1 and Sold it MessageID 75657
```

We run two buyers on elnux2 and two sellers on elnux3 and each buyer has two seller neighbors.

BuyerRouting.txt (elnux2, elnux3, elnux7)

```
Peer 3 Reply LookUp from peer 1 for product 1 MessageID 95657
Peer 3 received BUY from peer 1 for product 1 and Sold it MessageID 95657
```

We run two buyers on elnux2 and elnux3 and a seller on elnux7. Elnux3 is the neighbor of elnux2 and elnux7 is the neighbor of elnux3 so elnux3 can serve as mid node to route the LOOKUP message from elnux2 to elnux7.

```
Peer 3 Reply LookUp from peer 1 for product 1 MessageID 95657
Peer 3 received BUY from peer 1 for product 1 and Sold it MessageID 95657
```

BuyerAndSellerRouting.txt (elnux2, elnux3, elnux7)
Similar to the configuration in BuyerRouting.txt, we change role of the mid node from Buyer to BuyerAndSeller.

```
Peer 3 Reply LookUp from peer 1 for product 1 MessageID 95657
Peer 3 received BUY from peer 1 for product 1 and Sold it MessageID 95657
```
or
```
Peer 2 Reply LookUp from peer 1 for product 1 MessageID 95657
Peer 2 received BUY from peer 1 for product 1 and Sold it MessageID 95657
```
or
```
Peer 3 Reply LookUp from peer 2 for product 1 MessageID 95657
Peer 3 received BUY from peer 2 for product 1 and Sold it MessageID 95657
```

FullyConnected.txt