CSEE W4119 Programming HW 1

L I Yan(ly2278) ly2278@columbia.edu

Table of Contents

1 Announcement

The server terminal uses SQLite as the database to store item information and off-line message. SQLite is an open-source embedded SQL database engine (http://www.sqlite.org/).

The program is written in Java, with JDBC to link to the SQLite databse (http://code.google.com/p/sqlite-jdbc/).

2 Build Instruction

I introduce SQLite for server, thereby, we will have to run server by adding classpath.

```
cd ~/sobs/
make
```

for server:

```
java -classpath ".:sqlite-jdbc-3.7.2.jar" UdpSobs -s 2278
```

(**PS:** or directly run shell script by `sh server.sh`)

for client:

```
java UdpSobs -c 127.0.0.1 2278
```

(**PS:** or `java -classpath ".:sqlite-jdbc-3.7.2.jar" UdpSobs -c 127.0.0.1 2278`, or directly run shell script by `sh client.sh`)

3 Document Detail

The project core class is in the folder Terminal/, where there are "`Server.java`" and "`Client.java`".

`sobs/`

- `Terminal/`
  - `Server.java`
  - `Client.java`
  - `Operater.java`
  - `Listener.java`
- `Accessory/`
  - `Accessory.java`
  - `Bid.java`
  - `Buy.java`
  - `Sell.java`
- `Transmission/`
  - `ClientTransmission.java`
  - `ServerTransmission.java`
  - `TransmissionAddress.java`
- `srever.db`
- `UdpSobs.java`

- `makefile`
- `sqlite-jdbc-3.7.2.jar`
- `server.sh`
- `client.sh`

I will give a brief description of what these files are used for and how to use them in Section 5 and Section 6.

4 Program Feature

4.1 Server Main Features

1. registration & deregistration form clients.
2. updating & broadcasting clients' IP and Port.
3. listing items to clients.
4. dealing with clients' sell.
5. dealing with clients' bids.
6. dealing with clients' purchase.
7. informing sellers when their items are sold.
8. maintain off-line message for off-line or lost clients.
9. dealing with lost clients(see section 6).
10. input error control(see section 6).

4.2 Client Main Features

1. registration & deregistration.
2. asking server to list items.
3. selling items.
4. bidding an item.
5. buying an item.
6. dealing with server's address updating message.
7. receiving off-line message.
8. input error control(see section 6).
9. dealing with server lost.

5 Algorithm and Data Structure

5.1 Data Structure

5.1.1 Database

Server is stateble in my project, which means even server quite or force to quite by “ctrl+c”, all information about items and off-line messages will not lose. This is managed by a database system in my project. The database is in file `server.db`.

There are two tables in `server.db`: 1) items; 2) messages:

No	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Attribute	id	sellername	itemname	translimit	bidby	currentbid	buynow	desc
Type	INTEGER PK	VARCHAR(64)	VARCHAR(64)	INTEGER	VARCHAR(64)	INTEGER	INTEGER	VARCHAR(1024)

Table 5-1 items

No	*1*	*2*
Attribute	username	msg
Type	VARCHAR(64)	VARCHAR(1024)

Table 5-2 messages

The class to deal with the database is “`Database/serverDB.java`”: which mainly provides two methods: 1)`int sqlite_Execute(String)`; 2) `ResultSet sqlite_Query(String)`.

`sqlite_Execute` is for SQL querys that does not need return(e.g. inster, update...), `sqlite_Query` on the other hand is for SQL query which requires returns(e.g. select).

This class is used by server.

5.1.2 Store Network address

Both server and client will maintain a local memory which stores IP and prot for current existing clients. I use HashMap in Java to store it.

The HashMap field is in “`Transmission/ServerTransmission.java`” and “`Transmission/ClientTransmission.java`” for server and client respectively, which is `HashMap<String, TransmissionAddress>`. The key for the HashMap is client name and the value of it is class TransmissionAddress in “`Transmission/TransmissionAddress.java`”. Class TransmissionAddress has two fields: IP and Port.

5.1.3 Message format

All message transmitted between server-client, client-client have following format that: “*#xxx”. '*#' is the message header where '*' tells what kind of message it is. 'xxx' is the string which is sent.

The types of message are listed in emun in “`Transmission/ServerTransmission.java`” and

“`Transmission/ClientTransmission.java`”: `enum SockType {ACK, REGISTER, DEREGISTER, BROADCAST, SELL, INFO, BID, SOLD, BUY, DIRECT, PURCHASE, OFFLINE, ERROR}`. In this project, the receive classify the receiving data according to the message type.

In class Transmission, there are three methods which deals with message type: 1) `String SetTypeHeader(String, SockType)`; 2) `SockType GetTypeHead(String);` 3) `String RemoveTypeHeader(String)`, which play the role as their method name indicates.

5.2 Server

The main file for server is “`Terminal/Server.java`”, the basic flowchart for server is given in Fig 5-1:

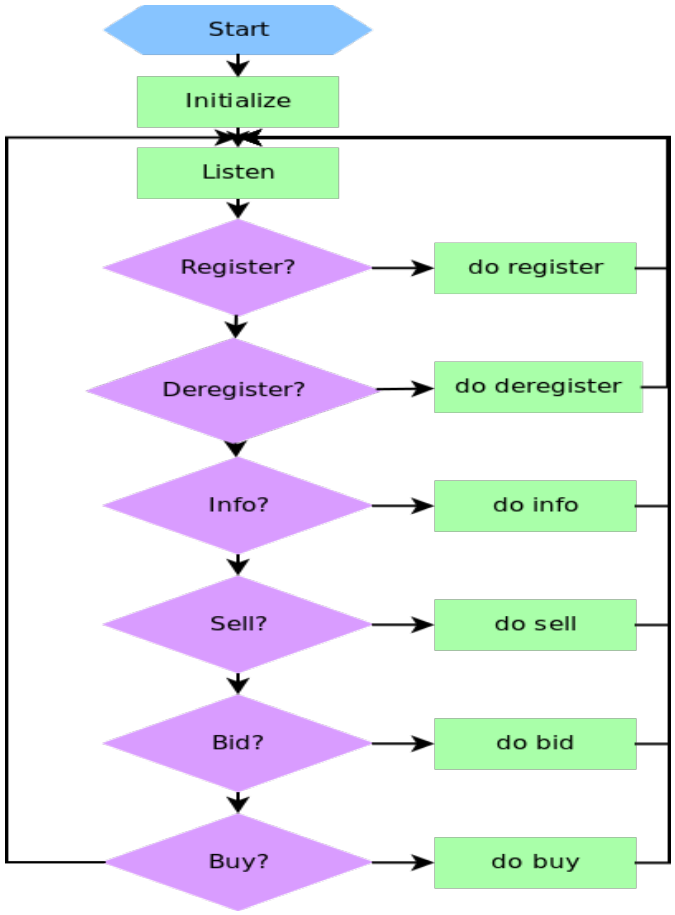

Fig 5-1 Server Flowchart

5.2.1 Register

see `Server.doRegister`

1. Reply ACK.
2. Put this client into addressHashMap.
3. For each address in the addressHashMap(existing users).
    - Broadcast the new addresses.

4. Select in database for table “messages” to see whether there are off-line messages to send to this client. If yes, send, otherwise continue. Whenever send a message, delete this message in the database.

Step 4 is done in `ServerTransmission.BroadcastAddr()`.

5.2.2 Deregister

see `Server.doDeregister`

1. Reply ACK.
2. Remove the client from the addressHashMap.
3. Broadcast the new addresses.

5.2.3 Sell

see `Server.doSell`

1. Parse the seller's message to see whether there is arguments mistake.
   - If argument mistake, send Error, return.
2. Get the seller's item information.
3. Insert the new item into the database.
4. Get the item's id from the database.
5. Set the reply message to seller (include the item id).

5.2.4 Info

see `Server.doInfo`

1. First select this item in the database according to the item id.
   - If item does not exist, send Error message, return.
   - If item exists, prepare item information message and send to client.

5.2.5 Bid

see `Server.doBid`

1. Parse the client's message to see whether there is arguments mistake.
   - If argument mistake, send Error, return.

2. Select the item in the database to get its sellername, itemname, translimit, bidby, currentbid.
    - If item not find, send Error message.
    - If sellername is the client, send Error message.
    - If bidby is the client, send Error message.
3. Then update the database by translimit -1, change bidby name, change currentbid.
    - If translimit is 0, the client has just got the item.
        1. Delete this item in the database. S
        2. Send purchase message to the client.
        3. Send sold message to seller. If seller is off-line. Update the database to store this message as an off-line message.

5.2.6 Direct Buy

see `Server.doDirect`

1. Select the item in the database to get itemname, buynow, sellername.
    - If item not find, tell Error to the sender.
    - If this direct buy message is send by seller, and if the item is not sold by this client, send Error both to seller and buyer.
    - If this direct buy message is send by buyer, and if the item is sold by this client, send owner Error message to this buyer.
2. Delete this item in the database,
3. Send purchase message to buyer (deal with lost client and off-line message).
4. Send sold message to seller (deal with lost client and off-line message).

5.3 Client

Client uses multi-threads, which is running two threads: 1) Operator; 2) Listener.

The basic flowchart for operator is like Fig 5-2(a) and flowchart for listener is like Fig 5-2(b).

The two threads are running individually see "`Terminal/Operater.java`" and "`Terminal/Listener.java`".

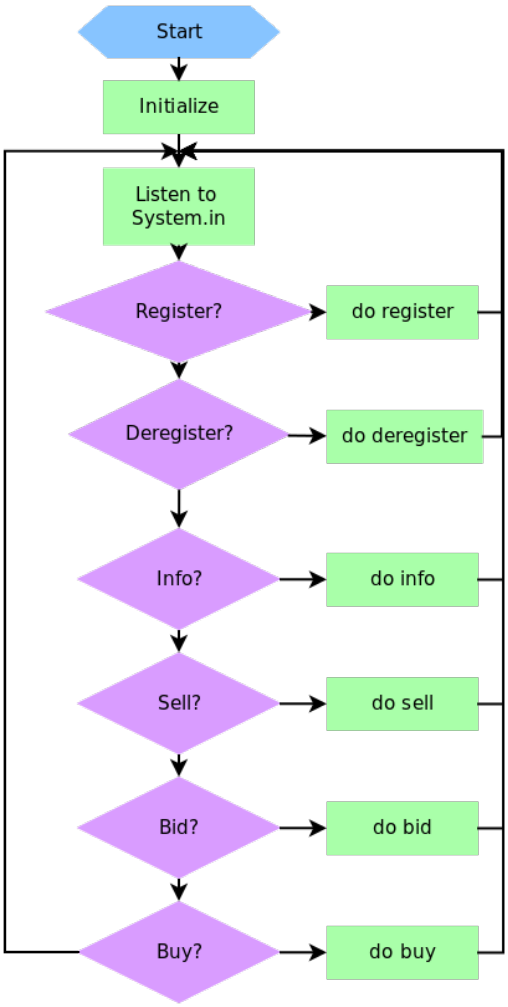

Fig 5-2(a) Operator Flowchart

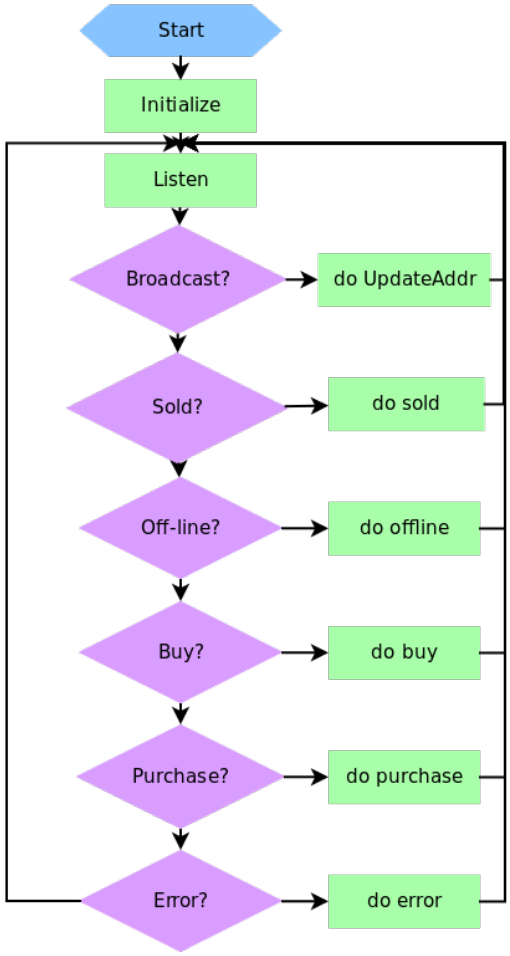

Fig 5-2(b) Listener Flowchart

5.3.1 Operator

All methods in operator are almost the same that it get user input from `System.in`, and check the input and send to server.

There is **one method is a different**, that is buy:

see `Operater.doBuy`

1. Parse input from the user.
2. Search the seller in the addressHashMap.
   - If the seller is in the addressHashMap, send buy message to the seller.
     - If send success return.
3. Here, either seller is not in the addressHashMap or it is off-line. Then send direct buy message to the server.

5.3.2 Listener

All methods in listener are almost the same that it listens to receive sockets, and print it **except broadcast and buy**.

see `Listener.doUpdateAddr`

This method calls `ClientTransmission.UpdateAddr`, this method parse the receiving message from the server and update it into addressHashMap.

see `Listener.doBuy`

1. Send buyer ACK.
2. Send direct buy message to server.

5.4 Transmission

Transmission methods are the most important methods in this project.

Mainly, there are two kinds of transmissions in my project: 1) transmission that requires ACK (e.g. Register message); 2) transmission that does not require ACK (e.g. ACK message).

5.4.1 Transmission that requires ACK

The method for transmission that requires ACK is method `Send_By_Address` both for "`ServerTransmission.java`" and "`ClientTransmission.java`". It processes as follows:

1. Prepare the data and DatagramPacket.
2. DatagramSocket.setSoTimeout(500).
3. If not receive ACK message, retry 5 times.
4. DatagramSocket.setSoTimeout(0).

Step 4 is to set the timeout back to 0, because it should block at receive in other situation.

5.4.2 Transmission that does not require ACK

This method just sends without doing something else.

6 Error control

Here I just demonstrate the result of my error control because some of them are done be many different methods both in client and server methods.
I also did error control to some additional potential errors as described below.

6.1 Input Error Control:

Input check is the file "`Accessory/Accessory.java`".

My input control supports any whitespaces between two arguments. Detail sees "**Test/input_test.txt**".

6.2 Lost client:

When server send message to client that requires ACK (e.g. Tell client that his/her item has been sold), if server timeout 5 times, it have to determine that the client is off-line, then the server have to remove the client address in the addressHashMap and broadcast to all users, see Fig 6-1:

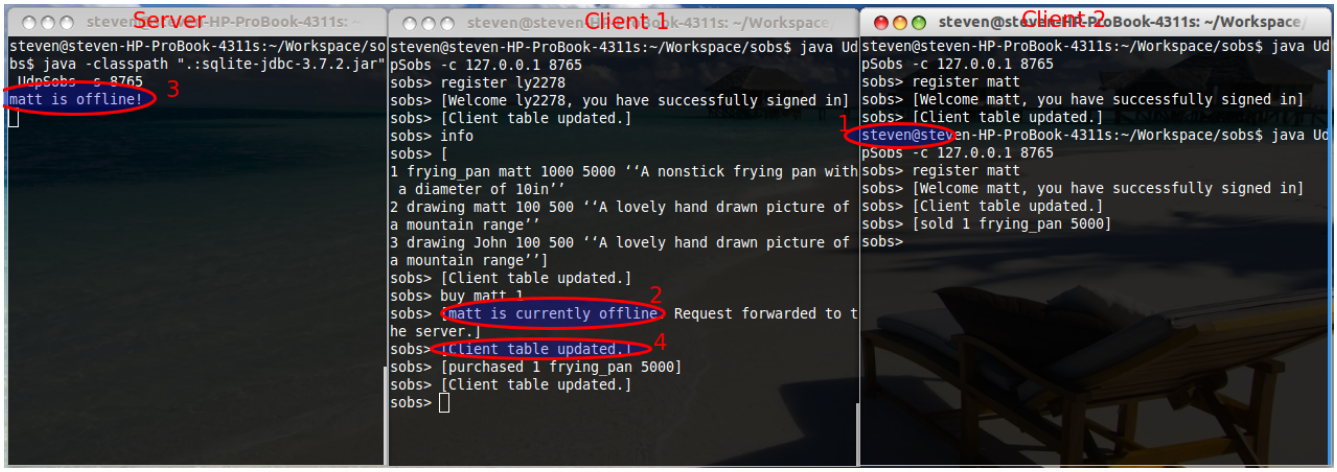

Fig 6-1 Lost client

In step 1: I stop client 2(matt) by "Ctrl+C".

In step 2: Client 1(ly2278) buys an item to this matt (matt is stopped by me). It finds matt in the addressHashMap, and sends buy message to matt. But matt is stopped by me. After timeout, client 2 sends direct buy message to server.

In step 3: Server proved this trade. But after sending sold message to seller (client 1—matt), server will timeout. Server knows that **matt is lost**, it updates the addressHashMap and broadcasts it.

In step 4: Client 2 receives the broadcast message and updates the addressHashMap.

In this way, server updates current clients' address and broadcasts it whenever a client is lost.

6.3 Lost Server

Server might also be lost. If a client finds that the server is lost, it cancels current operation and marks itself as "haven't signin".

Marking itself as "haven't signin" is necessary because at that time the client has no idea whether the server remembers him/her or not. Thereby, he/she has to register again. See Fig 6-2:

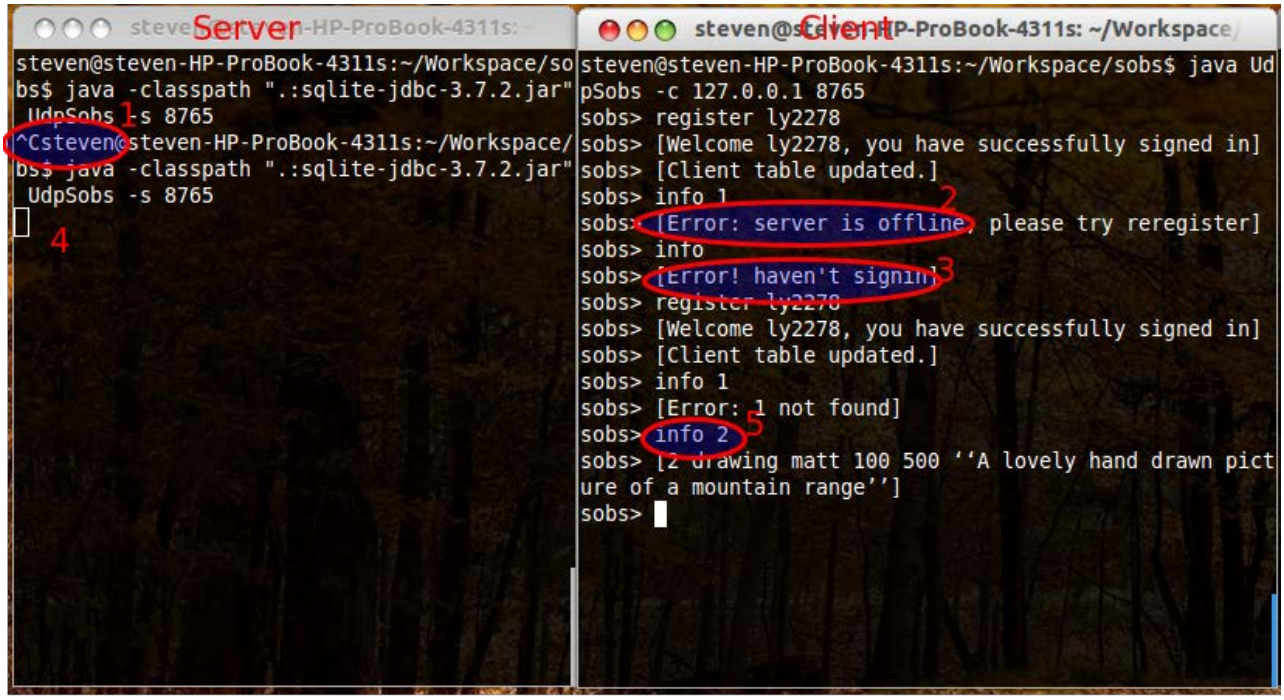

Fig 6-2 Lost Server

After server and client start:

In step 1: I stop the server by “Ctrl+C”.

In step 2: The client asks for info, but timeout. The server is lost. Then he/she marks him/her as signout.

In step 3: The client tries to do ask info, but the program tells him/her that he/she is now signout, he/she has to register again.

In step 4: I start the server.

In step 5: After the client register again, he/she can do normal operation with the server.

6.4 Buy from wrong seller:

There is a **potential error** as follows:

Buyer: `buy John 2`

Suppose John is on-line. In this way, buyer sends buy message to John and John sends direct buy message to server. However, John might not be the seller of this item.

In this way, the server had better cancel this trade and inform the seller and buyer.

In Fig 6-3:

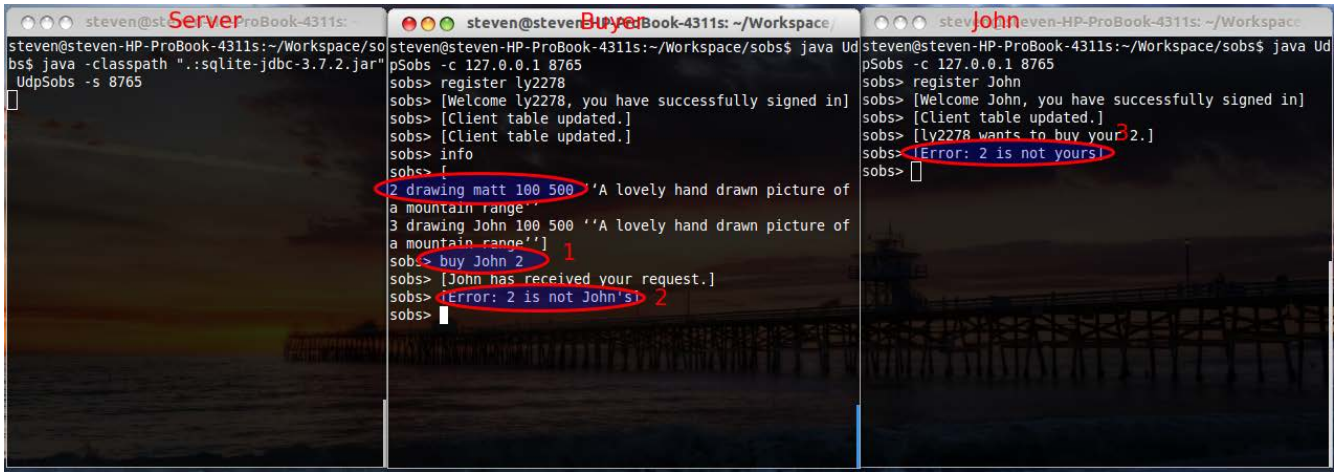

Fig 6-3 Buy from wrong seller

After the server and the buyer and John start. We see that item 2 is sold by matt.

In step 1: the buyer type: `buy John 2`. Then he/she send the buy message to John. John sends it to the server.

In step 2: the server searches the database and finds that the item is not John's. It sends Error to buyer and John. Then the buyer receives the Error message.

In step 3: John receives the Error message.

6.5 Buy his/her own item:

Another **potential error** as follows:

Buyer: `buy John 2`

And this time suppose John is off-line, after timeout, buyer would send direct buy to server, when the server finds the buyer is just the seller. It should send Error message to the buyer. See Fig 6-4:

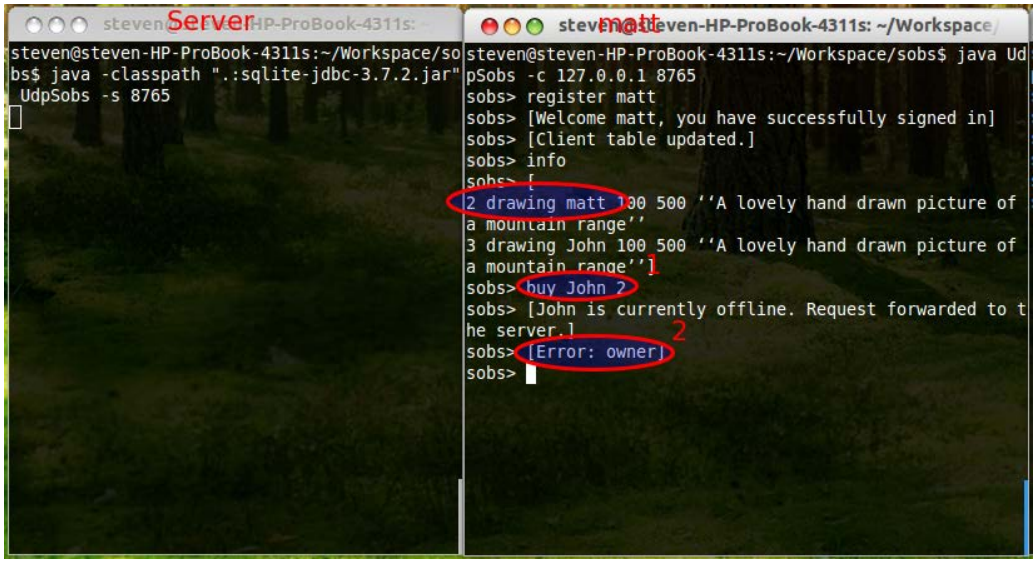

Fig 6-4 Buy his/her own item

We see item 2 is matt's.

In step 1: matt input: `buy John 2`. After timeout, matt send direct buy to server.

In step 2: server searches the database, and it finds that this item is sold by matt. Therefore, matt cannot buy it. It send Error message to matt.

7 Test

I simple test is given in file "**Test/simple_test.txt**". My project support multi-clients, which means there can be many on-line clients at the same time.

8 Gratitude

Thanks for TAs' great help to me for my project.