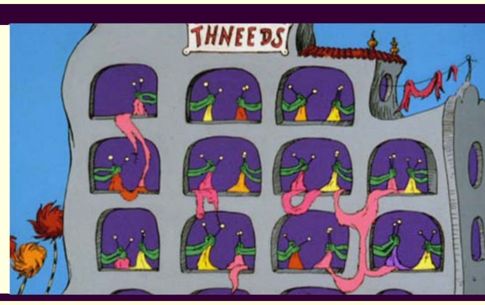
CS 351
Design of Large Programs
Java and Socket Communication

Instructor:
Joel Castellanos

e-mail: joel@unm.edu



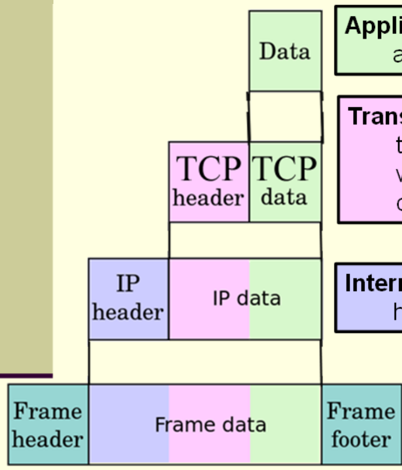
4/6/2017

Transmission Control Protocol

- The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite (IP).
- TCP provides reliable, ordered and error-checked delivery of a stream of octets between programs running on computers connected to a local area network, intranet or the public Internet.
- Web browsers use TCP when they connect to servers on the World Wide Web, and it is used to deliver email and transfer files from one location to another.
- HTTP, HTTPS, SMTP, POP3, IMAP, SSH, FTP, Telnet and a variety of other protocols are typically encapsulated in TCP.

2

TCP/IP Abstraction Layers



Application Layer where the higher level protocols such as SMTP, FTP, SSH, HTTP, etc. operate.

Transport Layer: responsible for end-to-end message transfer independent of underlying network, along with any error control, segmentation, congestion control & application addressing (via port numbers).

Internet layer: Host addressing and identification via hierarchical IP addressing system and packet routing

Link layer: protocols used to describe the local network topology and the interfaces needed to effect transmission of Internet layer datagrams to next-neighbor hosts without intervening routers.

3

What is a Socket?

- TCP (Transmission Control Protocol) provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other.
- To communicate over TCP, a client program and a server program establish a connection to one another.
- Each program binds a socket to its end of the connection.
- To communicate, the client and the server each reads from and writes to the socket bound to the connection.
- A socket is one end-point of a two-way communication link between two programs running on the network.

4

TCP Connection Establishment

TCP uses a three-way handshake:

- Server must first bind to and listen at a port. This is called a ***passive open***.
- Once the passive open is established, a client may initiate an ***active open***.

SYN: Client sends SYN to server. The client sets the segment's sequence number to a random value, A.

SYN-ACK: Server replies with acknowledgment number set to one more than the received sequence number (A+1), and another random number, B.

ACK: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value (A+1), and the acknowledgement number is set to one more than the received sequence number (B+1).

5

User Datagram Protocol (UDP)

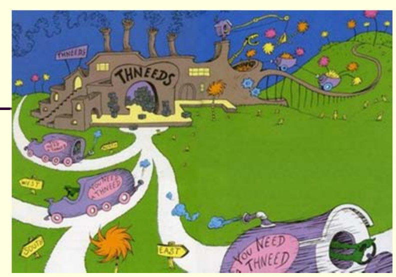
- When more speed is needed for data being transferred using TCP, can some speed be gained by using UDP instead?
- I would tell you a UDP joke, but you might not get it.

6

Client/Server Socket Connections

- Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.
- A prospective client must know the hostname of the machine on which the server is running and the port number on which the server is listening.
- To make a connection request, the client needs to identify itself to the server (hostname and local port number).
- If the server accepts the connection, the server creates a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket (running on a new thread) so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.
- On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.
- Client and server can now write to or read from their sockets.

Thneed Store



- Everyone needs a Thneed.
- Isn't about time we created a way to buy and sell Thneeds?
- For this lab we will be creating a client/server program that uses socket communication to track the purchase and sales of Thneeds.
- The server is the central office/warehouse.
- Each client is a remote sales / buyer representative of the central office.

Four Public Classes

ServerMaster: Manages connections. There is only 1 instance of this class.

ThneedStore: Manages Inventory. There is only 1 instance of this class.

ServerWorker: Manages communications with 1 client. There is 1 instance of ServerWorker per client.

Client: Connection between user and ServerWorker. There is 1 instance per user.

Partly working versions of 3 of these classes are available on the website. Feel free to use, modify or rewrite these as you see fit.

9

`ServerMaster.java`

The ***ServerMaster*** is the server side entry point (main) and runs in a single thread.

The ***ServerMaster*** instantiates a ThneedStore. Then it listens for new clients requesting to be added to the network. Since it runs in a single thread, it cannot do anything else while listening for a new client.

The ***ServerMaster*** creates a ServerWorker thread for each client connection request.

The ***ServerMaster*** must have a public method that the ThneedStore can call to broadcast a message to all clients. Therefore, the ***ServerMaster*** must maintain a list of connected clients. Since clients may connect and disconnect, **`java.util.LinkedList`** is a good structure for this.

10

ThneedStore.java

The ***ThneedStore*** tracks the inventory of Thneeds.

The ***ThneedStore*** is not a thread. Rather, it has **synchronized** accessor methods which can be called by any of the ServerWorker threads.

Each accessor ***must complete a full transaction before returning***.

If a client requests to sell more than the current inventory, then the request fails.

If a client request fails, the accessor must send a denial message to the client.

If a client request succeeds, the accessor must call a method in the ServerMaster to broadcast the updated inventory.

11

ServerWorker.java

ServerWorker **`extends`** **`Thread`**.

Each ***ServerWorker*** thread listens for requests from its client (to buy or sell Threeds).

Each ***ServerWorker*** thread blocks while listening for a message from its client.

When a ***ServerWorker*** thread receives a buy or sell request from its client, the ***ServerWorker*** calls a accessor method in the ThneedStore.

If the request is successful, the ***ServerWorker*** thread returns to listening for messages from its client.

If the request failed, the ***ServerWorker*** forwards the fail message to its client, then returns to listening.

12

`Client.java`

The ***Client*** class is the client side entry point (main).

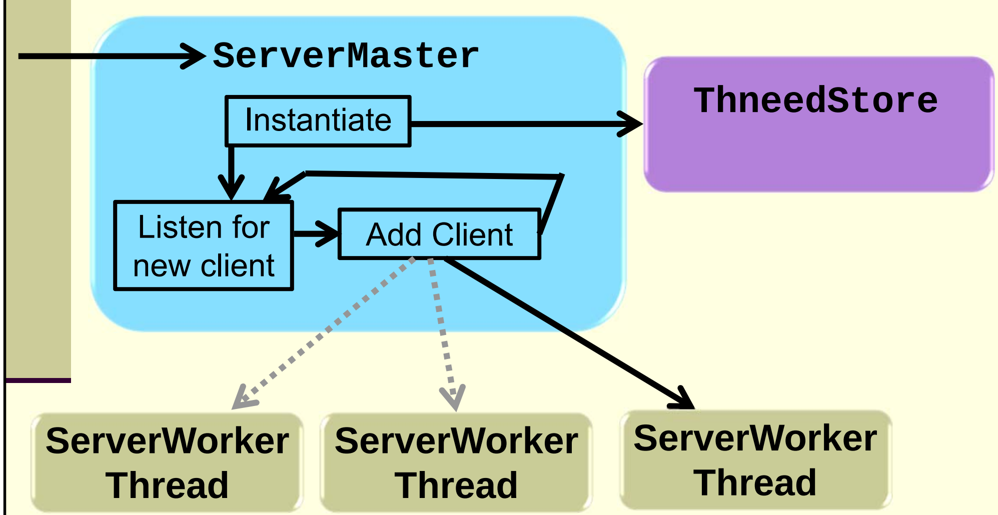
When ***Client*** starts, opens a socket connection to ServerMaster.

After connecting, ***Client*** must run in two threads:

1. One thread blocks while listening for input form the user (use **`java.util.Scanner`**).
2. The other thread blocks while listening to the socket connection it opened. Both ServerMaster and the thread's ServerWorker can send messages on this socket connection.

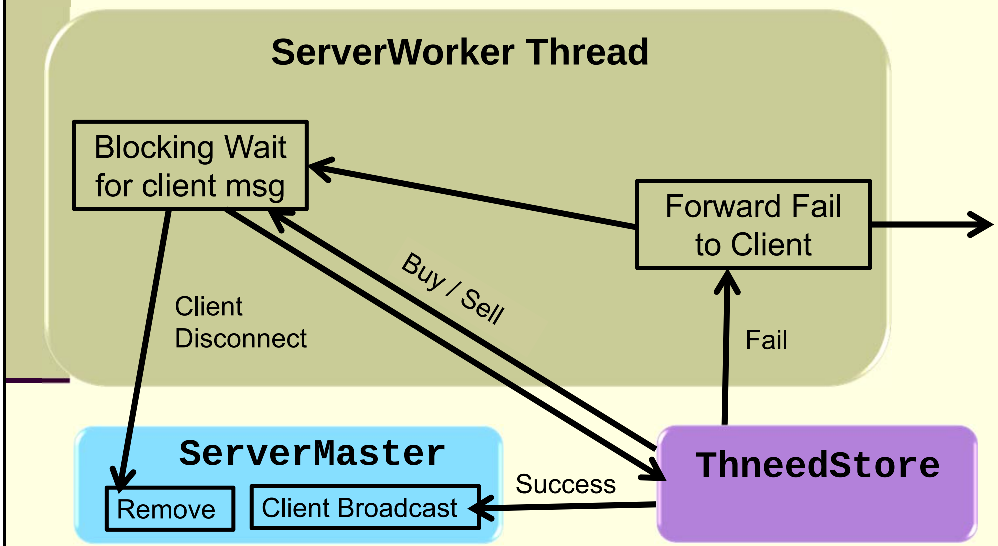
13

ServerMaster Main Thread



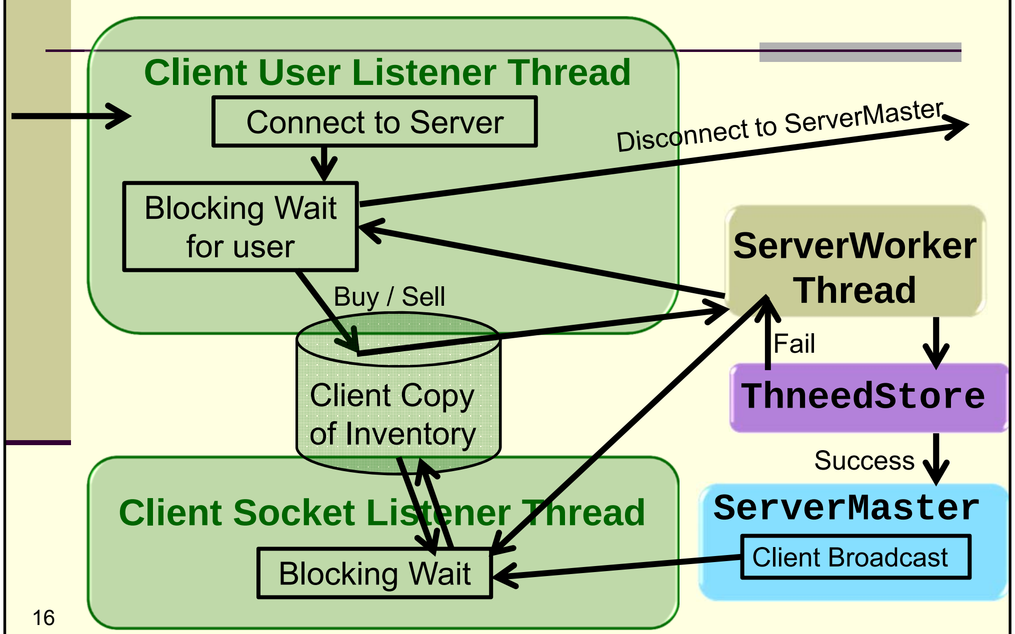
14

ServerWorker Thread



15

Client Threads



16

Initial State

At creation, the central office (thneedStore), has:

1) An inventory of zero thneeds.
2) A treasury balance of $1000 dollars.
3) No connected clients.

User Commands (1 of 2)

The client must accept and correctly process the following user commands when input at the console:

- **`buy:`** *quantity unitPrice*
- **`sell:`** *quantity unitPrice*
- **`quit:`**
- **`inventory:`**

quantity must be a positive integer.

price must be a dollar amount in the form *n.dd* where *n* is non-negative integer and each *d* is a digit character.

The **buy** command attempts to add *quantity* to the store's inventory and subtract *quantity*×*price* from the store's treasury. The **`sell`** command acts in reverse.

User Commands: EXAMPLE INPUT

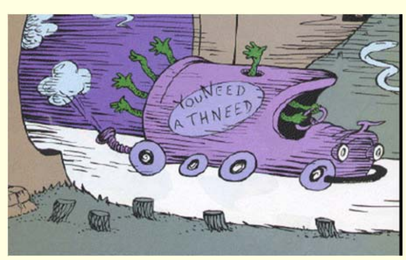
```
inventory:
buy: 30 10.23
buy: 50 9.10
sell: 20 18.95
buy: 100 8.12
sell: 10 20.25
sell: 15 21.95
inventory:
sell: 10 20.25
sell: 15 21.95
inventory:
quit:
```

19

User Commands (2 of 2)

The **`quit`** command must sent a disconnect message to the server and terminate the Client program. The server must not attempt to broadcast to client after it has received a disconnect message from that client.

The **`inventory`** Sends a message the server. It must display the result and update its local values.



20

Legal State of the Server

- Neither the Server's inventory nor its treasury may at any time between transactions be negative.
- All transactions must be atomic:
  1) Either all of a transaction is rejected or all of it is accepted.
  2) No transaction may begin reading or writing the state of the inventory or treasury while another transaction is in progress.

21

Server Broadcast Messages

- Immediately after each transaction, the server must broadcast an identical message to each client holding an open socket connection.
- The format and content of the message must be:

  time`: inventory=`*quantity* `: treasury=`*dollars*

  time must be the number of seconds, showing three decimal places, elapsed since the server first started through the end of the transaction.

  Each token in this broadcast message may be separated with zero or more whitespace characters.

  quantity: must be numeric characters that parse to an int.

  dollars: *d*`[`*d*`[`*d*`[`...`]]].`*dd*, where *d* is a numeric character.

22

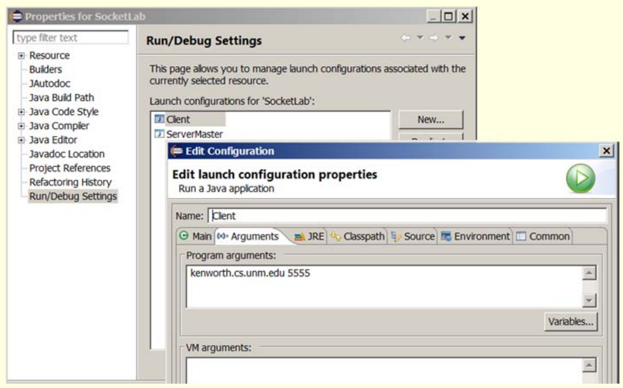
Eclipse: Setting Command Line Arguments

If you want to test from Eclipse, set the command line arguments for the Client by selecting:

Project → Properties → Run/Debug Settings → Client → Edit... → Arguments

and entering the command line arguments in the "Program arguments" field.

Then, do the same for the ServerMaster.



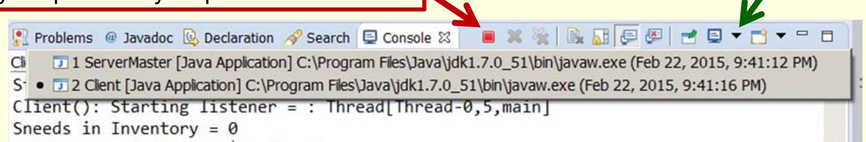
23

Running Two Processes in Eclipse

- For early testing, you may want to run both the server and the client on the same computer. To do this:
  1) Right-click on the ServerMaster class in the Navigator and select "Run as Java Application".
  2) Set the client's run configuration arguments to "localhost" and the same port number specified in the ServerMaster arguments. Then, Right-click on Client class in the Navigator and select "Run as Java Application".
- This will start two separate JVMs.
- All threads in a single JVM output to the same console.
- Different JVMs output to different consoles.

Click here to switch JVM consoles

Clicking "stop" will only stop the selected JVM



24

Grading Rubric

[-5 Points]: If the server does not run from the command line with the command:
`java ServerMaster` *portNumber*

[-5 Points]: If the client does not run from the command line with the command:
`java Client` *hostname portNumber*

[-5 Points]: If client or server displays a GUI or extraneous output.

[-5 Points]: If code doesn't adhere to the CS-351 coding standard. This includes adding comments as required in the standard to any instructor supplied code that you use.

[+10 Points]: Program passes slow speed command/response tests.

[+10 Points]: Program passes high speed command/response tests.

25

Turning in the Lab:

- Submit this assignment in Blackboard Learn as a .jar or .zip with the three .java files.
- Do NOT place your classes in a package.
- No need to create an executable .jar as they are being run by specifying **java ServerMaster** and **java Client** .
- Your code will be tested at HIGH SPEED (1000s of transactions per second with 3+ client instances each running on different machines). This is done as in CS-241, by redirecting a file to standard in. Your code will "think" it is reading from the keyboard, but it will actually get high speed input from a file.

26

Quiz

1) What is the Decorator Pattern?

2) Give a real example in the Java API where the Decorator Pattern is used.