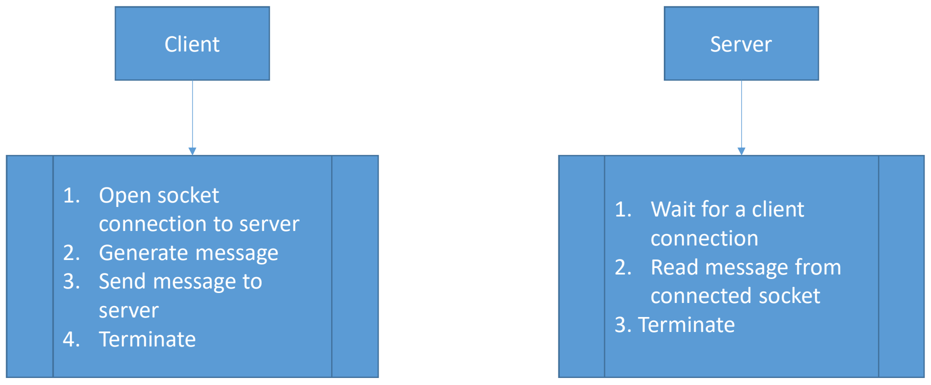
Client-Server Application Basic Chat

CSCI 2300

First client-server application



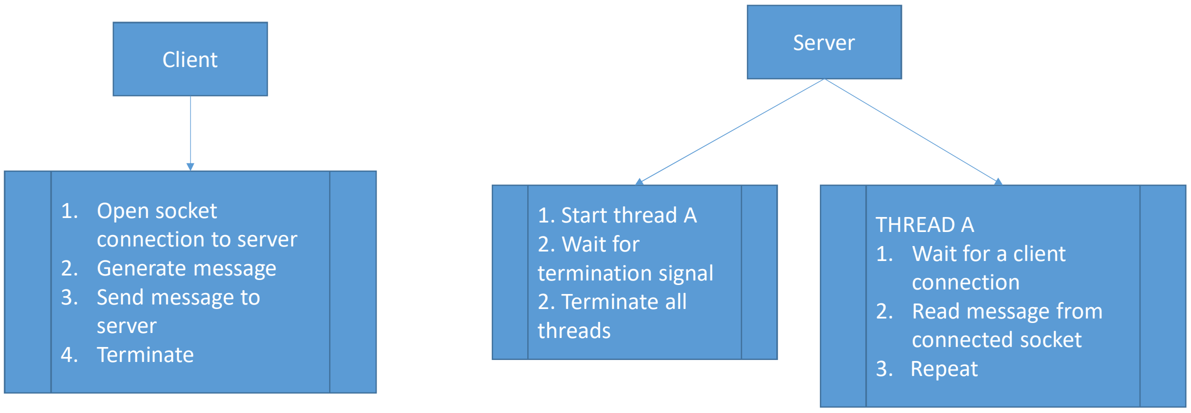
Client:

1. Open socket connection to server
2. Generate message
3. Send message to server
4. Terminate

Server:

1. Wait for a client connection
2. Read message from connected socket
3. Terminate

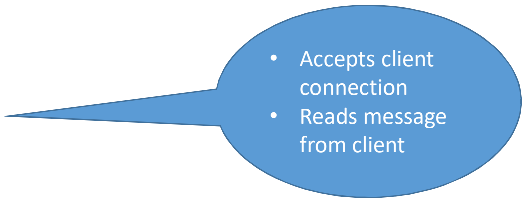
Modification: Server continues to accept client connections


Client
1. Open socket connection to server
2. Generate message
3. Send message to server
4. Terminate
Server
1. Start thread A
2. Wait for termination signal
2. Terminate all threads
THREAD A
1. Wait for a client connection
2. Read message from connected socket
3. Repeat

The `repeat` subdirectory

- TextMessage – unchanged
- Client – minor change: to run Client, you must provide <NAME> on command line: java -cp $CLASSPATH Kate
- Server – implements runnable (THREAD A on the previous diagram)

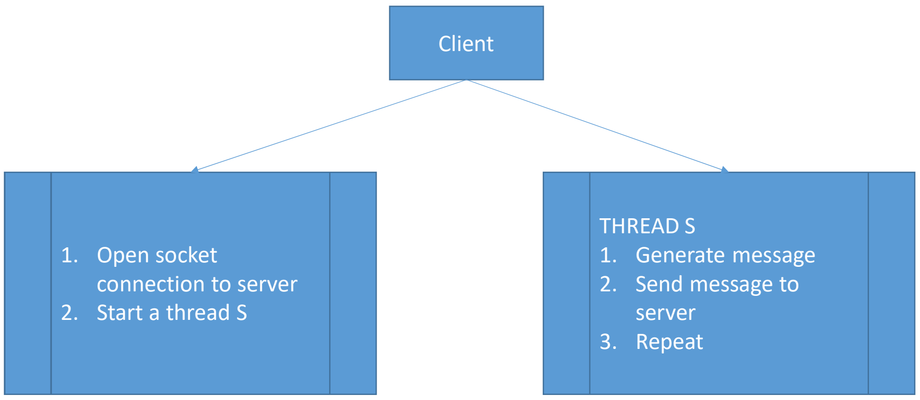
```
public void run()
{
    while (thread != null)
    {
        readTextMessage();
    }
}
```



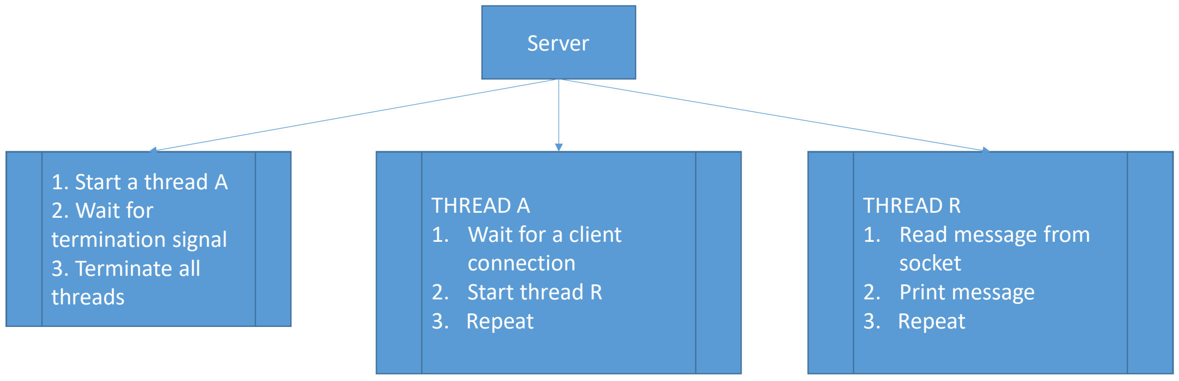
In `repeat` subdirectory, which of the following statements is true

A. A connected client can send multiple messages to the server
B. The server can accept multiple client connections, one at a time
C. The server can read messages from multiple clients in parallel
D. The server sends a response to a client, after receiving a message
E. All of the above

Modification: client can send multiple messages to the server



Modification: server can read messages from multiple clients in parallel

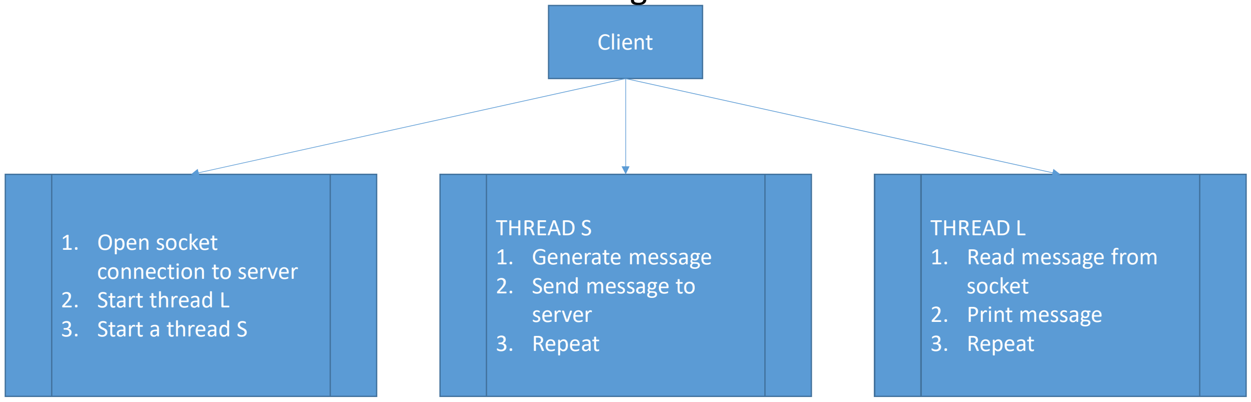


The `multi` subdirectory

- TextMessage - unchanged
- Client implements Runnable (THREAD S on the previous Client diagram)
- Server's run() method modified:
  - Repeatedly calls `acceptNewClient()`
  - `acceptNewClient()` accepts client connection and starts `ChatServerThread`
- ChatServerThread – new class (THREAD R on the previous Server diagram)
-
```
ssh –X hopper.slu.edu
cd <your git repo>/client_server/client_server
source ./configure.sh
cd multi
javac -cp $CLASSPATH *.java
java -cp $CLASSPATH Client <YOUR_NAME>
```

Basic Chat Application

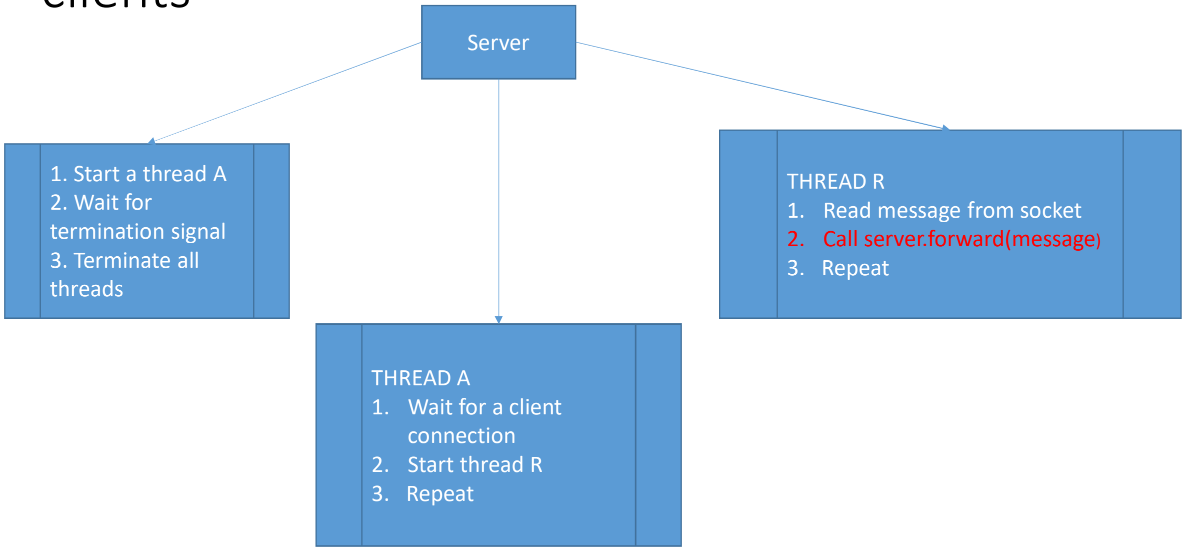
- Server – needs to forward messages to all connected clients
- Client – needs to “listen” for messages from the server



Basic Chat – Server side

- ChatServerThread – receives messages from one client
- Need to “forward” these messages to all clients
- Server has access to all client connections
  - ChatServerThread can pass the message to Server
  - Server can forward the message to all clients
  - ChatServerThread needs a reference to Server

Server forwards messages to connected clients


Server
1. Start a thread A
2. Wait for termination signal
3. Terminate all threads
THREAD A
1. Wait for a client connection
2. Start thread R
3. Repeat
THREAD R
1. Read message from socket
2. Call server.forward(message)
3. Repeat

Modifications in basic_chat

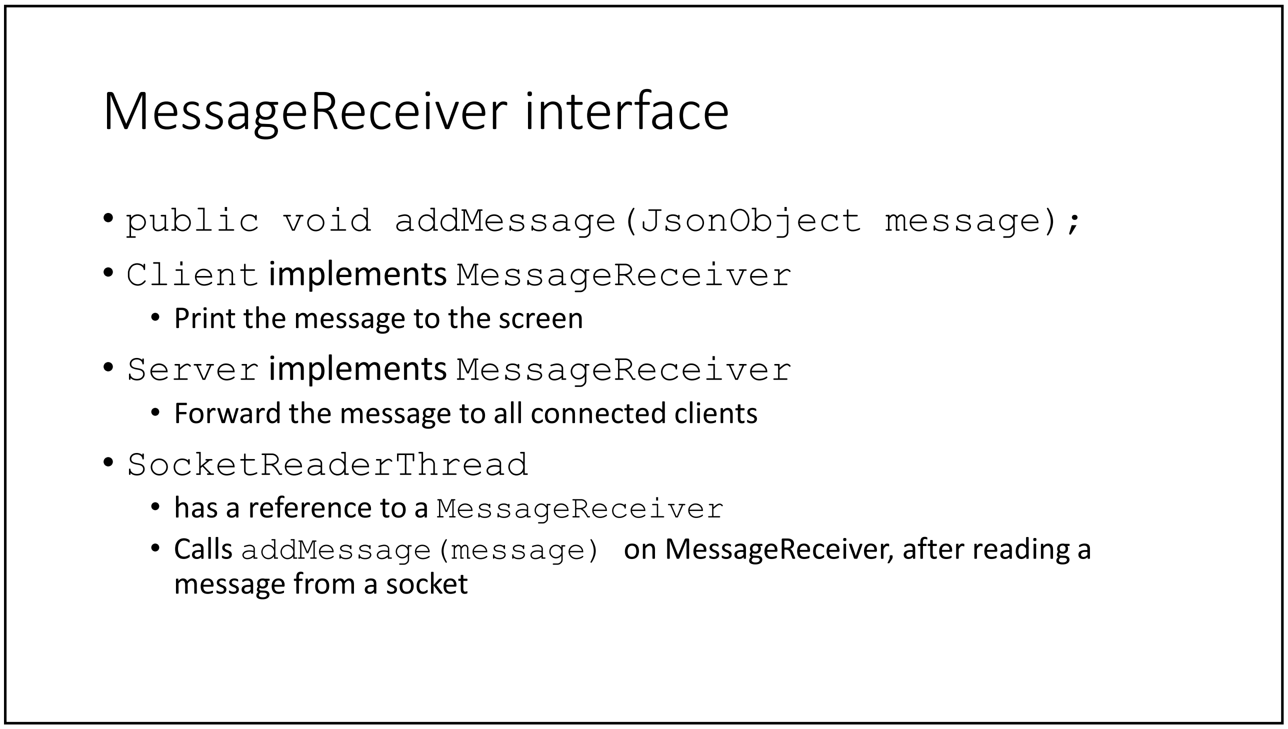
- Client starts thread L
- SocketReaderThread – new class (THREAD L)
- Server `forwardMessage(JsonObject message)` method added
- Added "locking" to ensure safe access to shared resource:
  - forwardMessage()
  - acceptNewClient()

What is the responsibility of SocketReaderThread?

A. Print messages to the screen
B. Read messages from a socket connection
C. Send messages to the server
D. Forward messages to other clients
E. All of the above

SocketReaderThread and ChatServerThread

- Responsibility: Read messages from a connected socket
- SocketReadereThread – prints received messages
- ChatServerThread – prints received messages and calls `server.forwardMessage()`
- Identical responsibility, two different implementations
- Can we combine them into one class?



MessageReceiver interface

- `public void addMessage(JsonObject message);`
- `Client` implements `MessageReceiver`
  - Print the message to the screen
- `Server` implements `MessageReceiver`
  - Forward the message to all connected clients
- `SocketReaderThread`
  - has a reference to a `MessageReceiver`
  - Calls `addMessage(message)` on MessageReceiver, after reading a message from a socket