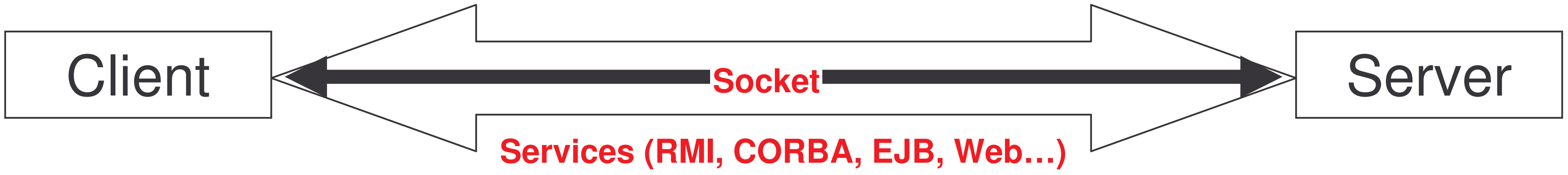
Socket

- Socket = communication channel



- Sockets in Java
  - Transparency (Unix: synchronous, Windows: asynchronous)
  - Complexity reduction (development)
  - Code compression
  - Client Socket ≠ Server Socket

Socket > Client/Server socket

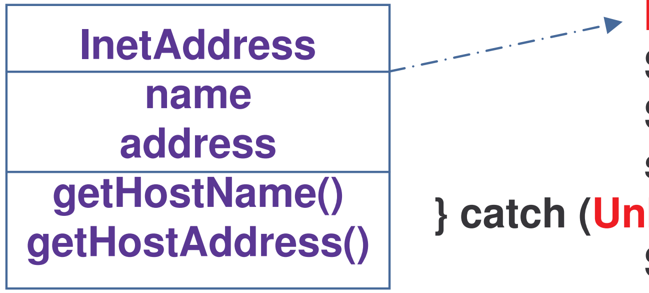
```
import java.net.*;
class Client {
        public static void main(String[ ] arg) {
                try {           Socket sk = new Socket("localhost", 9999);          // creation
                                /* Inputs and outputs */
                                sk.close();                                         // destruction
                } catch(java.io.IOException e) {
                                System.err.println("Connexion error on server: " + e);
                }
        }
}
class Server {
        public static void main(String[ ] arg) {
                try {           ServerSocket ssk = new ServerSocket(9999);          // creation
                                Socket skC;                                         // client
                                System.out.println("Server is running on: " + port);
                                while ((skC = ssk.accept()) != null) process(skC);
                } catch(java.io.IOException e) { System.err.println("Error on server: " + e); }
        }
        static void process(Socket sck) throws java.io.IOException {
                System.out.println("Connection of client: " + sck.getInetAddress());
                /* Inputs and outputs */
                sck.close();
        }
}
```

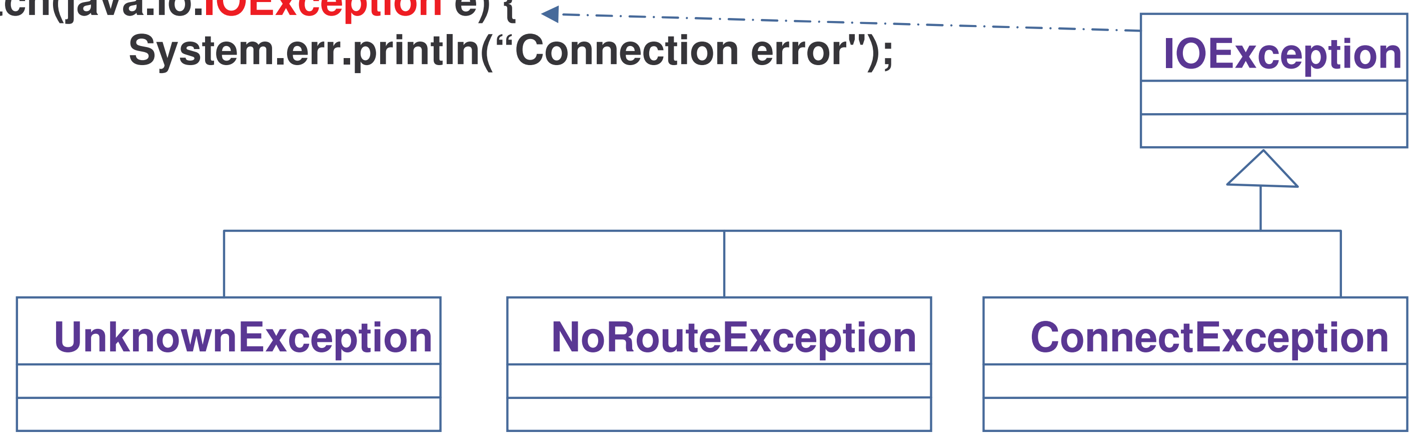
free port (>1024)

java.net.Socket() ⟹ getHostbyname(): find the server ⟹ socket(): configure sockaddr_in ⟹ connect()

Socket > Net address/error

```
import java.net.*;
class Client {
        public static void main(String[ ] arg) {
                try {
                        Socket sk = new Socket("localhost", 8080);
                        InetAddress svr = sk.getInetAddress();
                        System.out.println("Connected to " + svr.getHostName());
                        System.out.println(" on " + svr.getHostAddress());
                        sock.close();
                } catch (UnknownHostException e) {
                        System.err.println("Name error: unknown server");
                } catch (NoRouteToHostException e) {
                        System.err.println("Access error: server not reachable");
                } catch (ConnectException e) {
                        System.err.println("Access error: connection refused");
                } catch(java.io.IOException e) {
                        System.err.println("Connection error");
                }
        }
}
```



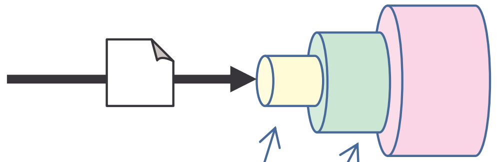


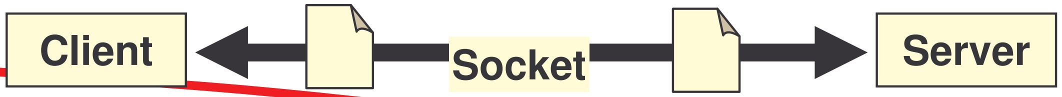
Socket > Data transfer

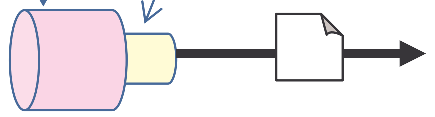
```
import java.net.*; import java.io.*; import java.util.*;
class Client {
            public static void main(String[ ] arg) {
                        try {       Socket sk = new Socket("localhost", 1951);
                                    InputStream is = sk.getInputStream();
                                    BufferedInputStream buf= new BufferedInputStream(is);
                                    ObjectInputStream ois = new ObjectInputStream(buf);
                                    Object o = ois.readObject();
                                    if (!(o instanceof Date))
                                    throw new IllegalArgumentException(o + "instead of Date");
                                    System.out.println("Today: " + ((Date)o).toString()); sk.close();
                        } catch(ClassNotFoundException e) { System.err.println("Invalid class");
                        } catch(java.io.IOException e) { System.err.println(e); }
            }
}
class Server {
            public static void main(String[ ] arg) {
                        try {       ServerSocket ssk = new ServerSocket(1951);
                                    while ((Socket skC = ssk.accept()) != null) process(skC);
                        } catch(java.io.IOException e) { System.err.println(" Server error: " + e); }
            }
            static void process(Socket sck) throws java.io.IOException {
                        ObjectOutputStream oos = new ObjectOutputStream(sck.getOutputStream());
                        oos.writeObject(new Date()); oos.close(); sck.close();
            }
}
```

for text tranfer, use:
InputStreamReader
BufferedReader
readLine()
write()

for object transfer

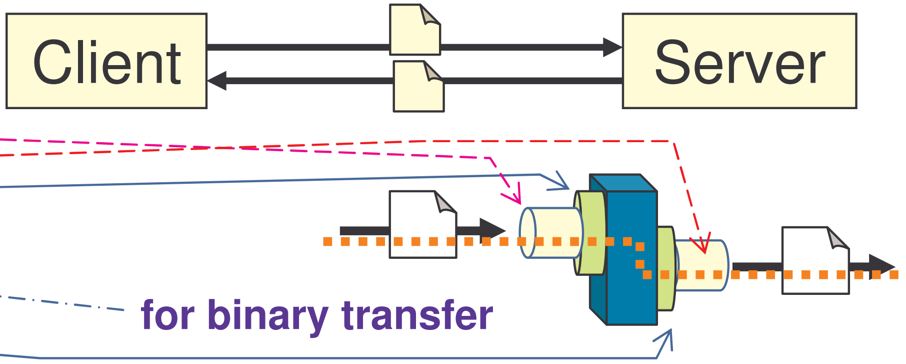




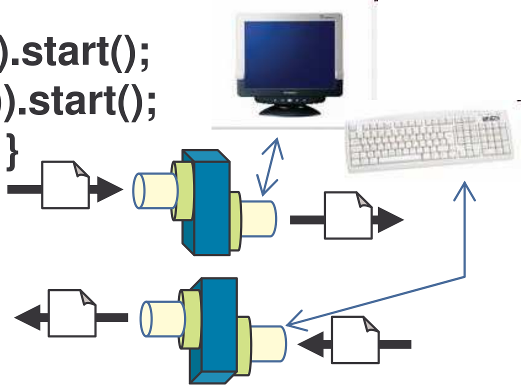


Socket > Parallel communication

```
import java.net.*; import java.io.*;
class Pipe extends Thread {
            DataInputStream dis;
            PrintStream ps;
            Pipe(InputStream is, OutputStream os) {
                        dis = new DataInputStream(is);
                        ps = new PrintStream(os);
            }
            public void run() {
                        try {       while((String st = dis.readLine()) != null) {
                                                ps.print(st); ps.print("\r\n"); ps.flush();
                                    }
                        } catch (IOException e) { throw new RuntimeException(e.getMessage()); }
            }
}
```



```
class TelnetClient {
            public static void main(String[ ] arg) {
                        try {       Socket sk = new Socket("localhost", 23);
                                    new Pipe(sk.getInputStream(), System.out).start();
                                    new Pipe(System.in, sk.getOutputStream()).start();
                        } catch(java.io.IOException e) { System.err.println(e); }
            }
}
```



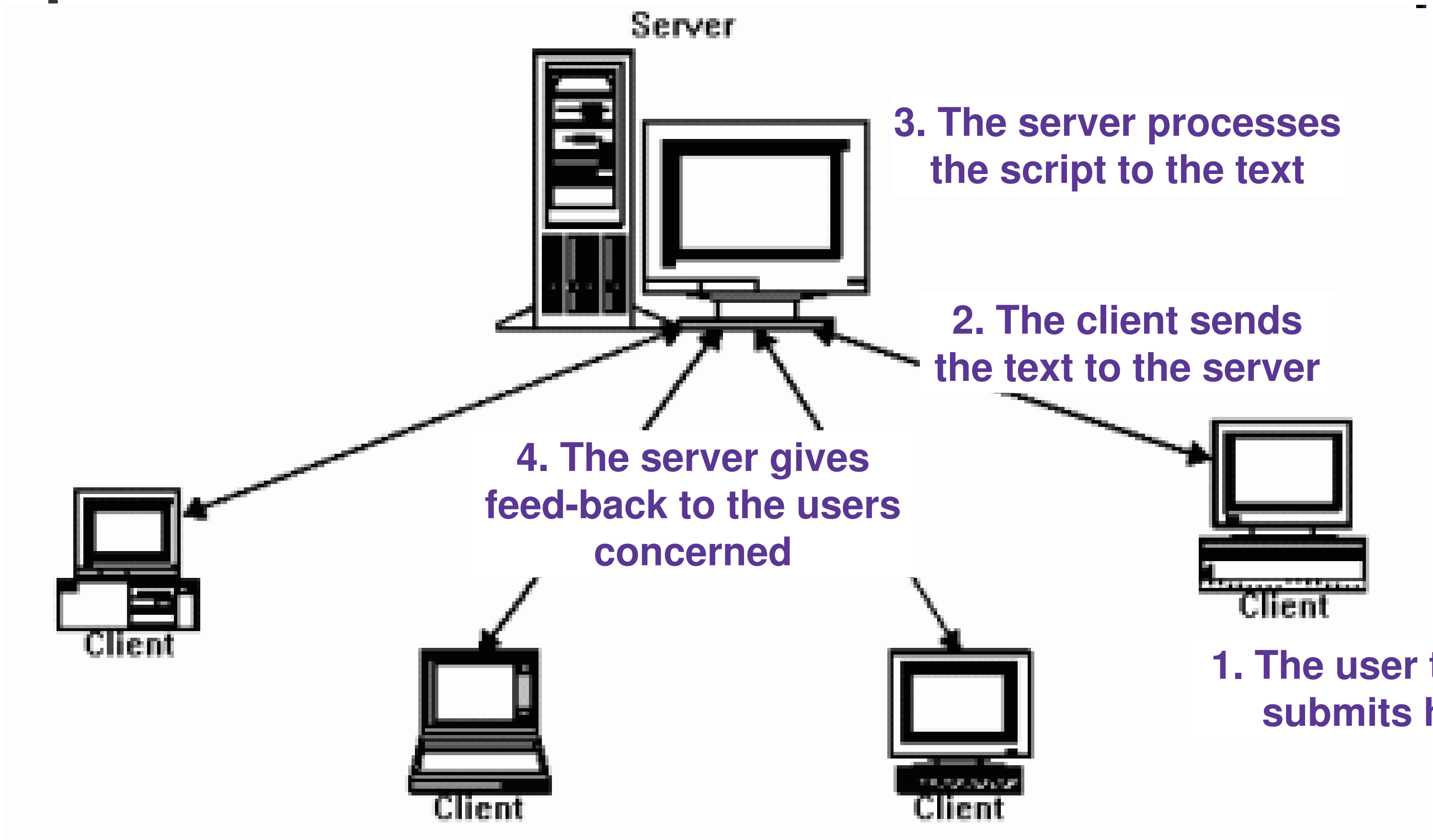
Socket > UDP protocol

M. K. Traoré

```
class UDP_Client {
        public static void main(String args[ ]) throws Exception {
                BufferedReader v = new BufferedReader(new InputStreamReader(System.in));
                DatagramSocket dsk = new DatagramSocket();
                byte[ ] s = new byte[1024], r = new byte[1024];
                String sentence = v.readLine(); s = sentence.getBytes();
                DatagramPacket dp = new DatagramPacket(s, s.length, 127.0.0.1, 4567);
                dsk.send(s);
                DatagramPacket dp = new DatagramPacket(r, r.length);
                dsk.receive(r); System.out.println("FROM SERVER:" + new String(r));
        }
}
public class UDP_Server {
        public static void main(String[ ] args) throws IOException {
                DatagramSocket dsk = new DatagramSocket(4567);
                byte[ ] b = new byte[1024];
                DatagramPacket r = new DatagramPacket(b,b.length);
                while(true) {
                        dsk.receive(r);
                        InetAddress source = r.getAddress(); int port = r.getPort();
                        String m = new String(b); byte[] s = m.toUpperCase().getBytes();
                        DatagramPacket s = new DatagramPacket(s,s.length,source,port);
                        dsk.send(sent);
                }
        }
}
```

Socket > IRC case study

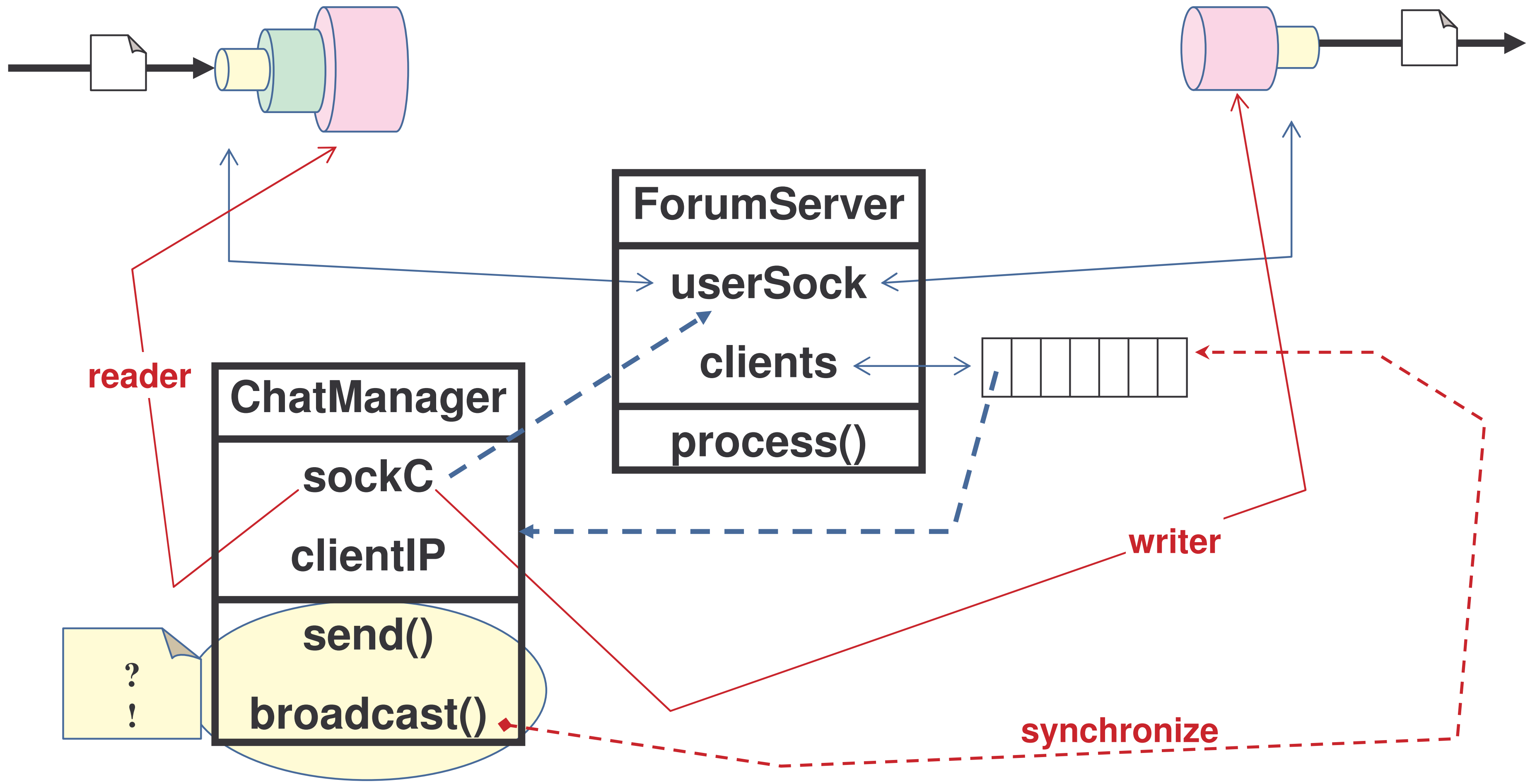
- Internet Relay Chat : enabler of text exchange within groups in real time


Server
3. The server processes the script to the text
2. The client sends the text to the server
4. The server gives feed-back to the users concerned
1. The user types and submits his text
Client
Client
Client
Client

- Code available here: client and server classes
- Deliverables: improved client and server classes

Socket > IRC server

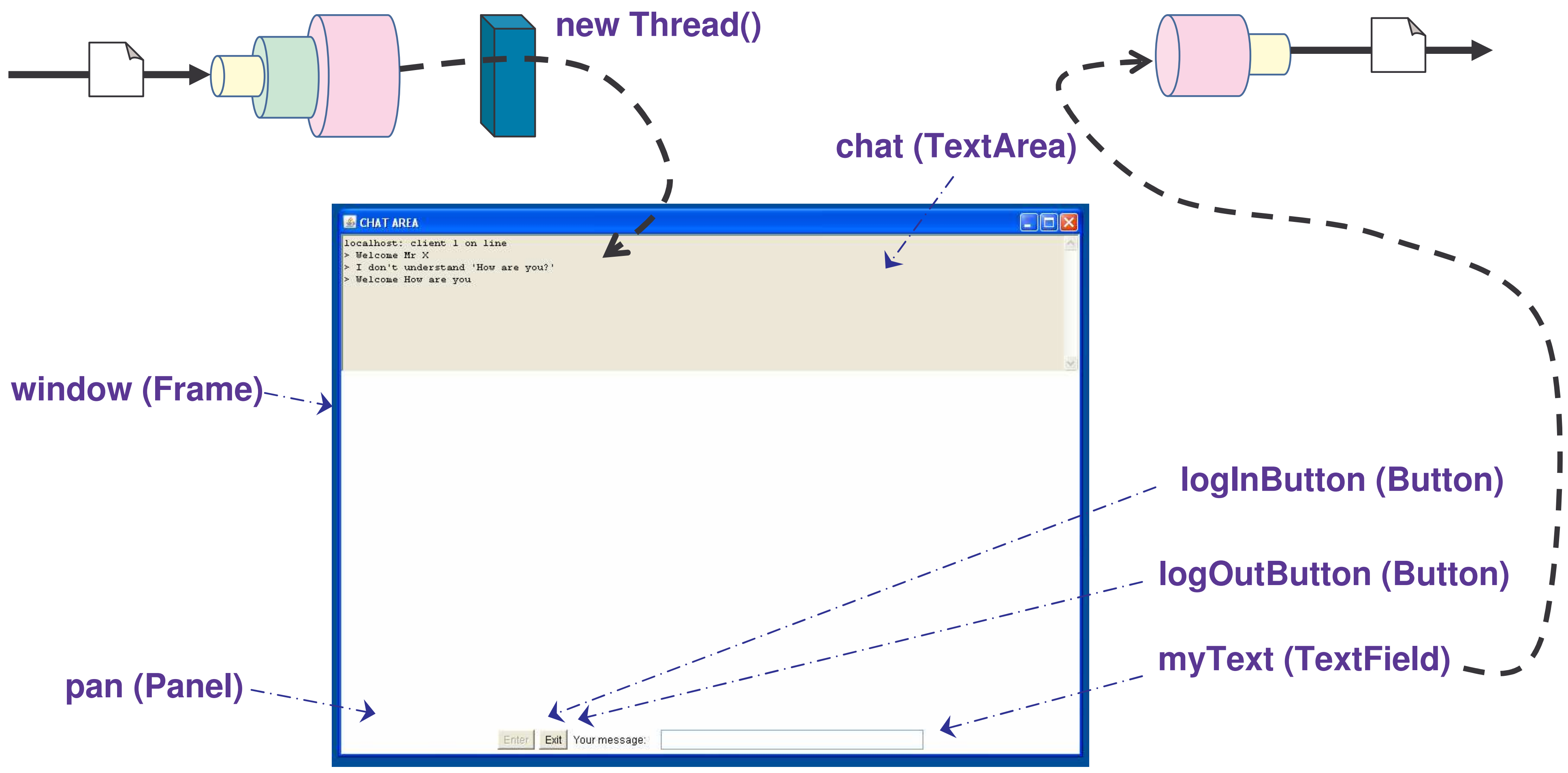
- Server class: involves an inner class (ChatManager)



Socket > IRC client

M. K. Traoré

- **Client class: Applet**



```
localhost: client 1 on line
> Welcome Mr X
> I don't understand 'How are you?'
> Welcome How are you
```

Socket > IRC improvements

- **Five improvements:**

1. **Any user is identified by a name (instead of "Mr. X")**
2. **The name is a key (used as prompt in messages)**
   - **logging without name ⇔ logging with name = "unknown"**
   - **"my_name" becomes "my_name2" if "my_name" already exists**
3. **A welcoming message is displayed to any incomer and a goodbye message is displayed to him when he leaves**
   - **all the users are advised of any arrival or departure and of the number of users currently connected to the system**
4. **Communication codes are:**
   - **! message: broadcast message headed by sender's name**
   - **@ name message: send message to the user identified by name**
   - **? name: rename the user with this new name and let all know**
   - **&: display help on communication codes**
   - **%: display the names of all the users that are currently connected**
5. **Error messages are sent for wrong code, name or number**