CSE3040 Java Language

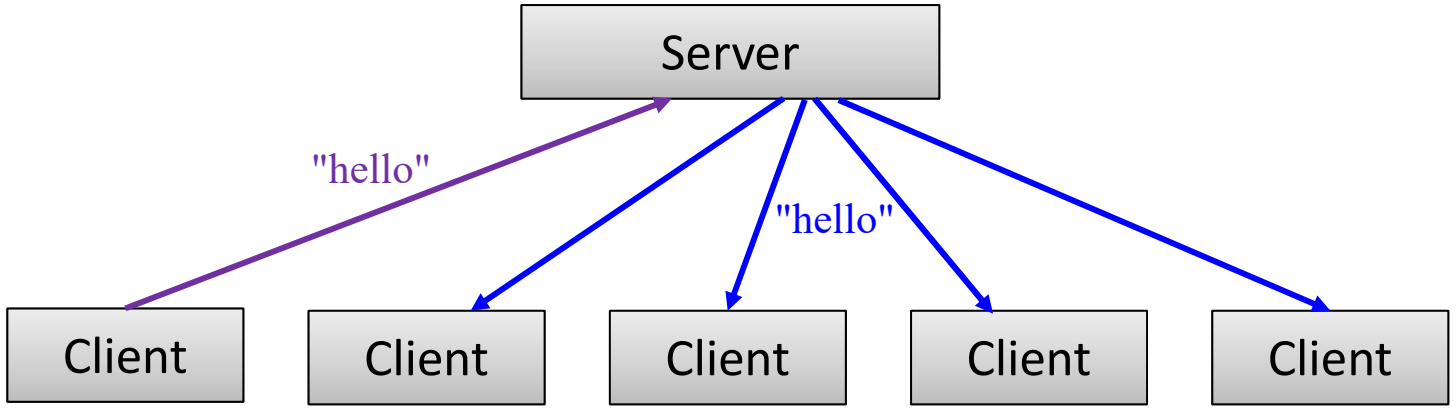
Lecture 25: Networking with Java (4)

Dept. of Computer Engineering,
Sogang University

Chat Program with Multiple Users

- MultiChatServer
  - Manages users
  - When a client sends a message, the server redirects the message to all other clients

- MultiChatClient
  - Connects to the server
  - Communicates with other clients that are connected to the server



Chat Program with Multiple Users

- MultiChatServer.java
  - uses a HashMap to store multiple clients. What is the problem of using a HashMap?

```
public class MultiChatServer {
    HashMap<String,DataOutputStream> clients;
    MultiChatServer() {
        clients = new HashMap<>();
        Collections.synchronizedMap(clients);
    }
    public void start() {
        ServerSocket serverSocket = null;
        Socket socket = null;
        try {
            serverSocket = new ServerSocket(7777);
            System.out.println("server has started.");
            while(true) {
                socket = serverSocket.accept();
                System.out.println("a new connection from [" + socket.getInetAddress() + ":" +
socket.getPort() + "]");
                ServerReceiver thread = new ServerReceiver(socket);
                thread.start();
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
```

Chat Program with Multiple Users

- MultiChatServer.java (cont.)
  - The sendToAll method iterates through the HashMap and sends data to all clients.
  - The list of clients includes the message sender, so the sender will also receive its own data from the server. How can you prevent that?

```
void sendToAll(String msg) {
     Iterator<String> it = clients.keySet().iterator();
     while(it.hasNext()) {
         try {
             DataOutputStream out = (DataOutputStream)clients.get(it.next());
             out.writeUTF(msg);
         } catch(IOException e) { }
     }
 }

 public static void main(String args[]) {
     new MultiChatServer().start();
 }
```

Chat Program with Multiple Users

- MultiChatServer.java (cont.)
  - class ServerReceiver is defined as a inner class.
  - The ServerReceiver is a thread, which has a DataInputStream and DataOutputStream.

```
class ServerReceiver extends Thread {
    Socket socket;
    DataInputStream in;
    DataOutputStream out;

    ServerReceiver(Socket socket) {
        this.socket = socket;
        try {
            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());
        } catch(IOException e) {}
    }
```

Chat Program with Multiple Users

- MultiChatServer.java (cont.)
  - When a new client comes, the ServerReceiver puts the new client into HashMap and notifies all clients that a new user has joined.
  - If the server receives data, it sends the data to all clients in the HashMap.
  - When a client is disconnected, the server notifies clients and removes the left client from the HashMap.

```
        public void run() {
            String name = "";
            try {
                name = in.readUTF();
                sendToAll("#"+name+" has joined.");
                clients.put(name, out);
                System.out.println("Current number of users: " + clients.size());
                while (in != null) {
                    sendToAll(in.readUTF());
                }
            } catch(IOException e) {
                // ignore
            } finally {
                sendToAll("#"+name+" has left.");
                clients.remove(name);
                System.out.println("["+socket.getInetAddress()+":"+socket.getPort()+"]"+" has disconnected.");
                System.out.println("Current number of users: " + clients.size());
            }
        }
    }
}
```

Chat Program with Multiple Users

- MultiChatClient.java
  - Class ClientSender is an inner class which implements a thread that sends data to the server.

```
public class MultiChatClient {
    static class ClientSender extends Thread {
        Socket socket;
        DataOutputStream out;
        String name;
        ClientSender(Socket socket, String name) {
            this.socket = socket;
            try {
                out = new DataOutputStream(socket.getOutputStream());
                this.name = name;
            } catch(Exception e) {}
        }
        @SuppressWarnings("all")
        public void run() {
            Scanner scanner = new Scanner(System.in);
            try {
                if (out != null) {
                    out.writeUTF(name);
                }
                while (out != null) {
                    out.writeUTF("["+name+"]"+scanner.nextLine());
                }
            } catch(IOException e) {}
        }
    }
```

Chat Program with Multiple Users

- MultiChatClient.java (cont.)
  - Class ClientReceiver is an inner class which implements a thread that receives data from the server.

```
static class ClientReceiver extends Thread {
    Socket socket;
    DataInputStream in;

    ClientReceiver(Socket socket) {
        this.socket = socket;
        try {
            in = new DataInputStream(socket.getInputStream());
        } catch(IOException e) {}
    }

    public void run() {
        while (in != null) {
            try {
                System.out.println(in.readUTF());
            } catch(IOException e) {}
        }
    }
}
```

Chat Program with Multiple Users

- MultiChatClient.java (cont.)
  - The main method creates a socket and connects to the server.
  - Then, it starts sender thread and receiver thread to communicate with the server.

```
    public static void main(String args[]) {
        if(args.length != 1) {
            System.out.println("usage: java MultichatClient username");
            System.exit(0);
        }

        try {
            String serverIp = "127.0.0.1";
            Socket socket = new Socket(serverIp, 7777);
            System.out.println("connected to server.");
            Thread sender = new Thread(new ClientSender(socket, args[0]));
            Thread receiver = new Thread(new ClientReceiver(socket));
            sender.start();
            receiver.start();
        } catch(ConnectException ce) {
            ce.printStackTrace();
        } catch(Exception e) {}
    }
}
```

UDP Socket Programming

- In UDP, no connection is established between a client and a server.

- A client or a server can create a DatagramSocket and send messages through the socket.
  - Message is sent to the destination by calling method send.
  - Naturally, there is no guarantee that the peer will receive the message.

- A client or a server can receive messages from a socket.
  - By calling method receive, the client or the server receives messages that is arrived at the socket.
  - The method receive is blocked until a message arrives at the socket.

UDP Socket Programming: Client

- The UdpClient creates a socket.
- Then, it sends a message to the server. The message size is 1 byte.
- Then, the UdpClient waits for incoming messages. It is blocked until it receives a message.

```
public class UdpClient {
    @SuppressWarnings("all")
    public void start() throws IOException, UnknownHostException {
        DatagramSocket datagramSocket = new DatagramSocket();
        InetAddress serverAddress = InetAddress.getByName("127.0.0.1");
        byte[] msg = new byte[100];
        DatagramPacket outPacket = new DatagramPacket(msg, 1, serverAddress, 7777);
        DatagramPacket inPacket = new DatagramPacket(msg, msg.length);
        datagramSocket.send(outPacket);
        datagramSocket.receive(inPacket);
        System.out.println("current server time: " + new String(inPacket.getData()));
    }

    public static void main(String args[]) {
        try {
            new UdpClient().start();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

UDP Socket Programming: Server

- The UdpServer first calls receive to wait for incoming messages.
- Once a message comes, the UdpServer sends a message containing current date to the client who sent the message. Methods getAddress() and getPort() is used to obtain client information.

```
public class UdpServer {
    @SuppressWarnings("all")
    public void start() throws IOException {
        DatagramSocket socket = new DatagramSocket(7777);
        DatagramPacket inPacket, outPacket;
        byte[] inMsg = new byte[10];
        byte[] outMsg;

        while(true) {
            inPacket = new DatagramPacket(inMsg, inMsg.length);
            socket.receive(inPacket);
            InetAddress address = inPacket.getAddress();
            int port = inPacket.getPort();

            SimpleDateFormat sdf = new SimpleDateFormat("[hh:mm:ss]");
            String time = sdf.format(new Date());
            outMsg = time.getBytes();
            outPacket = new DatagramPacket(outMsg, outMsg.length, address, port);
            socket.send(outPacket);
        }
    }
```

UDP Socket Programming: Server

- The main method starts the UdpServer.

```
   public static void main(String args[]) {
        try {
            new UdpServer().start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- In order to send a message through a UDP socket, we need to create a DatagramPacket instance, and send the DatagramPacket through DatagramSocket.
- When creating a DatagramPacket instance, the IP address and the port number of the destination should be specified.

- What will happen if we run the client first and the server later?

Programming Lab #25

25-01. ChatServer for Multiple Clients

- Execute the following code and understand the results.
- Create multiple clients and have them connect to the server.
- MultiChatServer.java

```
public class MultiChatServer {
    HashMap<String,DataOutputStream> clients;
    MultiChatServer() {
        clients = new HashMap<>();
        Collections.synchronizedMap(clients);
    }
    public void start() {
        ServerSocket serverSocket = null;
        Socket socket = null;
        try {
            serverSocket = new ServerSocket(7777);
            System.out.println("server has started.");
            while(true) {
                socket = serverSocket.accept();
                System.out.println("a new connection from [" + socket.getInetAddress() + ":" + socket.getPort() + "]");
                ServerReceiver thread = new ServerReceiver(socket);
                thread.start();
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
```

25-01. ChatServer for Multiple Clients

- MultiChatServer.java (cont.)

```
void sendToAll(String msg) {
       Iterator<String> it = clients.keySet().iterator();
       while(it.hasNext()) {
           try {
               DataOutputStream out = (DataOutputStream)clients.get(it.next());
               out.writeUTF(msg);
           } catch(IOException e) { }
       }
   }

   public static void main(String args[]) {
       new MultiChatServer().start();
   }

   class ServerReceiver extends Thread {
       Socket socket;
       DataInputStream in;
       DataOutputStream out;

       ServerReceiver(Socket socket) {
           this.socket = socket;
           try {
               in = new DataInputStream(socket.getInputStream());
               out = new DataOutputStream(socket.getOutputStream());
           } catch(IOException e) {}
       }
```

25-01. ChatServer for Multiple Clients

- MultiChatServer.java (cont.)

```
    public void run() {
            String name = "";
            try {
                name = in.readUTF();
                sendToAll("#"+name+" has joined.");
                clients.put(name, out);
                System.out.println("Current number of users: " + clients.size());
                while (in != null) {
                    sendToAll(in.readUTF());
                }
            } catch(IOException e) {
                // ignore
            } finally {
                sendToAll("#"+name+" has left.");
                clients.remove(name);
                System.out.println("["+socket.getInetAddress()+":"+socket.getPort()+"]"+" has disconnected.");
                System.out.println("Current number of users: " + clients.size());
            }
        }
    }
}
```

25-01. ChatServer for Multiple Clients

- MultiChatClient.java (cont.)

```
public class MultiChatClient {
    static class ClientSender extends Thread {
        Socket socket;
        DataOutputStream out;
        String name;
        ClientSender(Socket socket, String name) {
            this.socket = socket;
            try {
                out = new DataOutputStream(socket.getOutputStream());
                this.name = name;
            } catch(Exception e) {}
        }
        @SuppressWarnings("all")
        public void run() {
            Scanner scanner = new Scanner(System.in);
            try {
                if (out != null) {
                    out.writeUTF(name);
                }
                while (out != null) {
                    out.writeUTF("["+name+"]"+scanner.nextLine());
                }
            } catch(IOException e) {}
        }
    }
```

25-01. ChatServer for Multiple Clients

- MultiChatClient.java (cont.)

```
    static class ClientReceiver extends Thread {
        Socket socket;
        DataInputStream in;

        ClientReceiver(Socket socket) {
            this.socket = socket;
            try {
                in = new DataInputStream(socket.getInputStream());
            } catch(IOException e) {}
        }

        public void run() {
            while (in != null) {
                try {
                    System.out.println(in.readUTF());
                } catch(IOException e) {}
            }
        }
    }
```

25-01. ChatServer for Multiple Clients

- MultiChatClient.java (cont.)

```
    public static void main(String args[]) {
        if(args.length != 1) {
            System.out.println("usage: java MultichatClient username");
            System.exit(0);
        }

        try {
            String serverIp = "127.0.0.1";
            Socket socket = new Socket(serverIp, 7777);
            System.out.println("connected to server.");
            Thread sender = new Thread(new ClientSender(socket, args[0]));
            Thread receiver = new Thread(new ClientReceiver(socket));
            sender.start();
            receiver.start();
        } catch(ConnectException ce) {
            ce.printStackTrace();
        } catch(Exception e) {}
    }
}
```

25-02. Udp Server & Client

- Execute UdpServer first and then UdpClient.
  - What happens if you run UdpClient first?

UdpServer.java

```
public class UdpServer {
    @SuppressWarnings("all")
    public void start() throws IOException {
        DatagramSocket socket = new DatagramSocket(7777);
        DatagramPacket inPacket, outPacket;
        byte[] inMsg = new byte[10];
        byte[] outMsg;
        while(true) {
            inPacket = new DatagramPacket(inMsg, inMsg.length);
            socket.receive(inPacket);
            InetAddress address = inPacket.getAddress();
            int port = inPacket.getPort();
            SimpleDateFormat sdf = new SimpleDateFormat("[hh:mm:ss]");
            String time = sdf.format(new Date());
            outMsg = time.getBytes();
            outPacket = new DatagramPacket(outMsg, outMsg.length, address, port);
            socket.send(outPacket);
        }
    }
    public static void main(String args[]) {
        try {
            new UdpServer().start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

25-02. Udp Server & Client

- Execute UdpServer first and then UdpClient.
  - What happens if you run UdpClient first?

UdpClient.java

```
public class UdpClient {
    @SuppressWarnings("all")
    public void start() throws IOException, UnknownHostException {
        DatagramSocket datagramSocket = new DatagramSocket();
        InetAddress serverAddress = InetAddress.getByName("127.0.0.1");
        byte[] msg = new byte[100];
        DatagramPacket outPacket = new DatagramPacket(msg, 1, serverAddress, 7777);
        DatagramPacket inPacket = new DatagramPacket(msg, msg.length);
        datagramSocket.send(outPacket);
        datagramSocket.receive(inPacket);
        System.out.println("current server time: " + new String(inPacket.getData()));
    }

    public static void main(String args[]) {
        try {
            new UdpClient().start();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

End of Class

Instructor office: AS818A

Email: jso1@sogang.ac.kr