3. Program List

TEAM: Yifan Liu & Tianyang Che
NOTE: I will list some essential part of the program here.

1. Server connection:

IndexServer has a Serversocket listening on port 8101.
Then this server socket waits for a request.

```
ss = new ServerSocket(8101);
socket = ss.accept();
```

Server can return a socket to the client.
Client can easily get socket port.

```
Client.socket[i] = new Socket(Client.neighbors[i].peerIP,
        Integer.parseInt(Client.neighbors[i].peerPort));
```

2. Send content using input/output stream:

In our program, we use DataInputStream and DataOutPutStream to send message. The content in the message is typically a string.

```
dos = new DataOutputStream(socket.getOutputStream());
dis = new DataInputStream(socket.getInputStream());
```

To send message,

```
dos.writeUTF(sendBuffer);
```

To get message,

```
(dis.readUTF();
```

In order to convert hash map to a string, and then convert back, we uses gson to implement this functionality.

To convert from a class to a string,

```
String sendBuffer = gson.toJson(result);
dos.writeUTF(sendBuffer);
```

To convert from a string to a given class.

```
receivedHitMessage = gson.fromJson(dis.readUTF(),
        receivedHitMessage.getClass());
```

3. Multithread operation:

We use to implement runnable interface. Because there are two different kinds of thread, we need two classes, each of which implement the runnable interface.

```
public class Listener implements Runnable{

public class Handler implements Runnable
```

We need to override the run() function:

```
public void run() {
    try {
        String senderName;
        Client.serversocket = new ServerSocket(Integer.parseInt(Client.self.peerPort));
        while(true){
            socket = Client.serversocket.accept();
            dis = new DataInputStream(socket.getInputStream());
            senderName = dis.readUTF();
            //System.out.println("I receive from " + senderName);

            // Then, find the right slot in the static socket array.
            for(int i = 0 ; i < Client.neighborsNo ; i ++ ) {
                if (Client.neighbors[i].peerName.equals(senderName)) {
                    Client.socket[i] = socket;
                }
            }



        }
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Create a thread.

```
new Thread(new Handler(i)).start();
```

start() function is used to execute the run() function.

4. Manuel create network.

If a neighbor's socket is null, we create the socket. Otherwise, we don't do anything.

```
public void connect() {

    try {
        for (int i = 0; i < Client.neighborsNo; i++) {
            if (Client.socket[i] == null) {
                Client.socket[i] = new Socket(Client.neighbors[i].peerIP,
                        Integer.parseInt(Client.neighbors[i].peerPort));
                dos = new DataOutputStream(
                        Client.socket[i].getOutputStream());
                dos.writeUTF(Client.self.peerName);
            } else {
                // System.out.println(" Notice : "
                // + Client.neighbors[i].peerName + "has occupied.");
            }
        }

        while (this.hasConnected() == false) {
        }

        for (int i = 0; i < Client.neighborsNo; i++) {
            new Thread(new Handler(i)).start();
        }
        // System.out.println("Network established");

    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

5. Query.

When client A sends a query to search file F, it calls the query function. It firstly chooses the corresponding socket, and sends a command index "2" to tell the neighbor B 's correspond thread that A wants to send a message. And then send its own information to B as the upstream client.

```
public void query(MessageID mid, int TTL, String searchFileName) {
    Message m = new Message(mid, TTL, searchFileName);

    // send this message
    try {
        for (int i = 0; i < Client.neighborsNo; i++) {
            DataOutputStream dos = new DataOutputStream(
                    Client.socket[i].getOutputStream());
            dos.writeUTF("2");
            // send message
            Gson gson = new Gson();
            String sendBuffer = gson.toJson(m);
            dos.writeUTF(sendBuffer);
            // send upstream information
            sendBuffer = gson.toJson(Client.self);
            dos.writeUTF(sendBuffer);
        }
    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

As for the receiving client B, it firstly checks if it has this message m. If not, store the message and upstream client in the two global arrays.

```
            // if no, store the message in the array
            if (Client.checkMessageArray(temp) == false) {
                Client.messageArray[Client.messageNumber] = temp;
                Client.upstreamArray[Client.messageNumber] = p;
                Client.messageNumber++;
            }
```

Then, it forwards this message to all B's neighbors except the upstream.

```
if (temp.currentTTL != 0) {
    for (int i = 0; i < Client.neighborsNo; i++) {
        // not send to its upstream
        if (p.peerName.equals(Client.neighbors[i].peerName) == false) {

            DataOutputStream dos = new DataOutputStream(
                    Client.socket[i].getOutputStream());
            dos.writeUTF("2");
            // send message
            String sendBuffer = g.toJson(temp);
            dos.writeUTF(sendBuffer);
            // send upstream information
            sendBuffer = g.toJson(Client.self);
            dos.writeUTF(sendBuffer);
        }
    }
}
```

Then, B sends the hit message to the upstream. This will handle in the hit message part.

6. Hit query.

When a client gets a message, it must send a hit query message to the message's original sender along with the reverse path.

Suppose client A receives a message m, it's going to send a hit message h to the sender.

At first, A should check if it is m's sender. If yes, it doesn't need to send hit message to anybody. If no, A must send to A's upstream client.

```
HitMessage receivedHitMessage = new HitMessage();
receivedHitMessage = gson.fromJson(dis.readUTF(),
        receivedHitMessage.getClass());
// receivedHitMessage.printHitMessage();

// if current peer is not the sender, we continue sending
// this hit message to upstream.
int previousIndex = -1;
// 1. decide whether the current peer is the source.
if (receivedHitMessage.m.peerID.peerName
        .equals(Client.self.peerName) == false) {
    // 2. find upstream peer index
    for (int i = 0; i < Client.messageNumber; i++) {
        if (Client.messageArray[i].messageID
                .isEqual(receivedHitMessage.m)) {
            previousIndex = i;
        }
    }
}
```

If not, find m's upstream client.

```
// find the upstream client's socket index
if (previousIndex != -1) {
    for (int i = 0; i < Client.neighborsNo; i++) {
        if (Client.neighbors[i].peerName
                .equals(Client.upstreamArray[previousIndex].peerName))
            chooseSocket = i;
    }
}
```

Then, we need to check whether A has the search file and send the hit message to upstream client.

```
if (chooseSocket != -1) {
    dos = new DataOutputStream(
            Client.socket[chooseSocket].getOutputStream());

    dos.writeUTF("3");

    // 3. send back to the upstream peer.
    // public HitMessage(MessageID me, boolean fl, Peer
    // ta)
    // receivedHitMessage.target = Client.self;

    sendBuffer = gson.toJson(receivedHitMessage);
    dos.writeUTF(sendBuffer);
}
```

7. Obtain Operation:

Choose a file name, target client name, then send this information to the target client.

```
public void obtain(String fn) {
    System.out
            .println("Please select the peer name you want to download from :");
    Scanner obtainFile = new Scanner(System.in);
    String pn = obtainFile.nextLine();

    // 1. peek the socket belonging to the target peer.
    int socketIndex = -1;
    for (int i = 0; i < Client.neighborsNo; i++) {
        if (Client.neighbors[i].peerName.equals(pn))
            socketIndex = i;
    }
    try {
        System.out.println("socket index is " + socketIndex);
        // 2. send command index
        DataInputStream diss = new DataInputStream(
                Client.socket[socketIndex].getInputStream());
        DataOutputStream doss = new DataOutputStream(
                Client.socket[socketIndex].getOutputStream());
        doss.writeUTF("4");

        // 3. send file name
        doss.writeUTF(fn);
    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Target client send the file content back.

```
    // System.out.println("4");
    // 1. get file name
    String fn = dis.readUTF();
    // 2. search the name
    System.out.println("Download file is " + fn);
    String filePath = "/Users/yangkklt/cs550demo/"
            + Client.self.peerName + "/" + fn;
    // System.out.println(filePath);
    FileInputStream in = new FileInputStream(filePath);
    File fileTemp = new File("/Users/yangkklt/tempfile");
    FileOutputStream out = new FileOutputStream(fileTemp);
    int c;
    byte buffer[] = new byte[10240];
    // read the file to a buffer
    int textLength = 0;
    while ((c = in.read(buffer)) != -1) {
        for (int i = 0; i < c; i++) {
            out.write(buffer[i]);
            textLength = i;
        }
    }
    String str = new String(buffer, "UTF-8");
    // System.out.println(str);
    dos.writeUTF("5");
    dos.writeUTF(fn);
    dos.writeUTF(str);
```

Write to a local file.

```
System.out.println("command 5");
String fname = dis.readUTF();
String result = dis.readUTF();
result = result + "\0";

// write to local file
FileWriter fw = new FileWriter("/Users/yangkklt/cs550demo/"
        + Client.self.peerName + "/" + fname);
fw.write(result, 0, result.length());
fw.flush();
fw.close();
```