Advanced Operating Systems

CS-550

Programming Assignment 1

A Simple Napster Style Peer to Peer File Sharing System

Group Members:

Kunal Dhande

A20369862

kdhande@hawk.iit.edu

Mohit Kulpe

A20372614

mkulpe@hawk.iit.edu

Content Table

1. Introduction

1.1. Problem Statement

A simple P2P system is to be designed which has a central indexing server. The peers in the network will register the files on the indexing server. Central Indexing Server: The central indexing server will maintain an index of the files registers by peers. When a peer will make a request for file search it will return the peer ID for the peer who has register that file on the server. Peer: A peer will register the files on the indexing server to make them available for other peers. A peer will make a file search request to the indexing server and will get peer ID of the peer having that file. It will then connect to the peer to get the file.

1.2. System Introduction

This document provides a detailed design of the Distributed File Sharing System. It provides an overview of how different components in the system interact with each other.

Peer to Peer File Sharing System is a console based system which has centralized database (Indexing Server) where users can register their files. Other users can search files in database and download their respected files as per their requirement.

2. Architecture

The distributed file system has a central indexing server which indexes the files registered by the peers. The peers in the system register their files on the indexing server. The peers can also search for a file on the indexing server. The indexing server returns the information of peer on which that file is present. After getting information of peer the client connects to that peer and downloads the file.

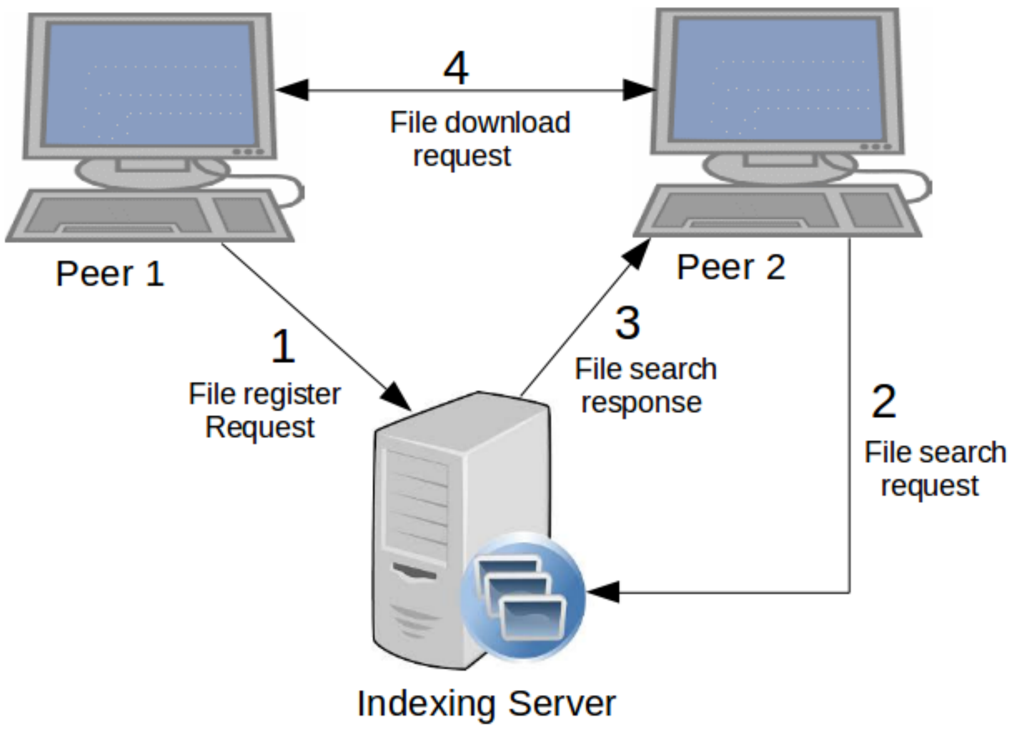


Figure 1

2.1 Working of system

1) Peer 1 sends a request to the indexing server to register the files it has, as shown in figure 1. It sends the list of file names, ip address and the port where it will be available for download on requests. On receiving request the indexing server makes entries for the files in its index table. It also stores the IP address and port number of the peer who has the files.

2) Peer 2 sends a search request to the indexing server, as shown in figure 1. It sends the file name to be searched in the index.

3) On receiving the search request Indexing server searches the file in the index. If file is found it returns the IP address and port number of all the peers who has the file as shown in figure 1.

4) Peer 2 then picks first peer from the list which he gets from indexing server in form of response, here Peer 1 has the file as shown in figure 1. So Peer 2 connects to Peer 1 and requests for that particular file, Peer 1 reads the file from its memory and sends the file to Peer 2.

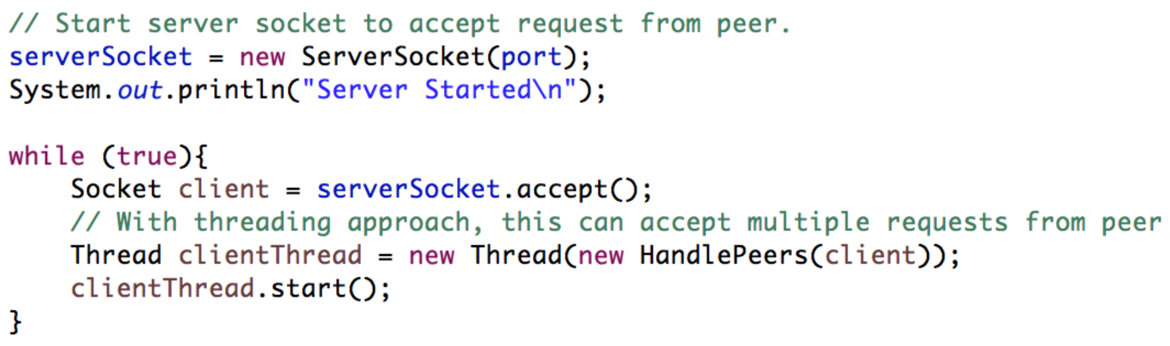
3. Implementation

3.1 Indexing Server

Indexing Server accepts the data from various Peer using Socket and Multithreading concepts. A new thread created on Indexing Server if client peer wants to register a file and this newly created thread is then taken over by the Indexing Server thread handler in order to fulfill Peer Client's request. The connection between Peer Client and Indexing Server is released once the operation is completed with either success or error.

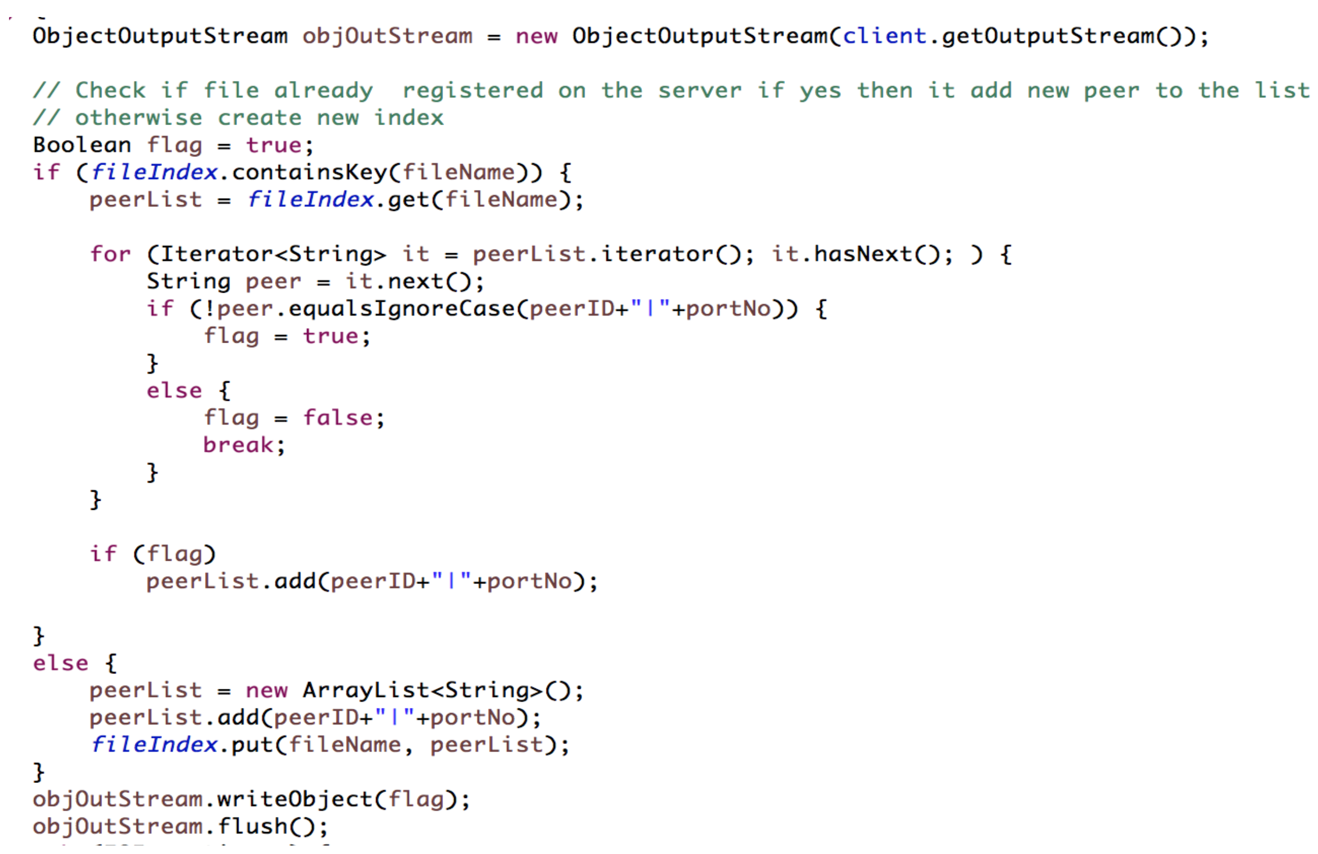
Indexing Server uses ConceurrentHashMap structure from Java to create an Indexing Server database. Operations carried out by indexing server are Register, Search, and Unregister based on the inputs received from Peer Client.

3.1.1 Concurrent Peer Request Handling



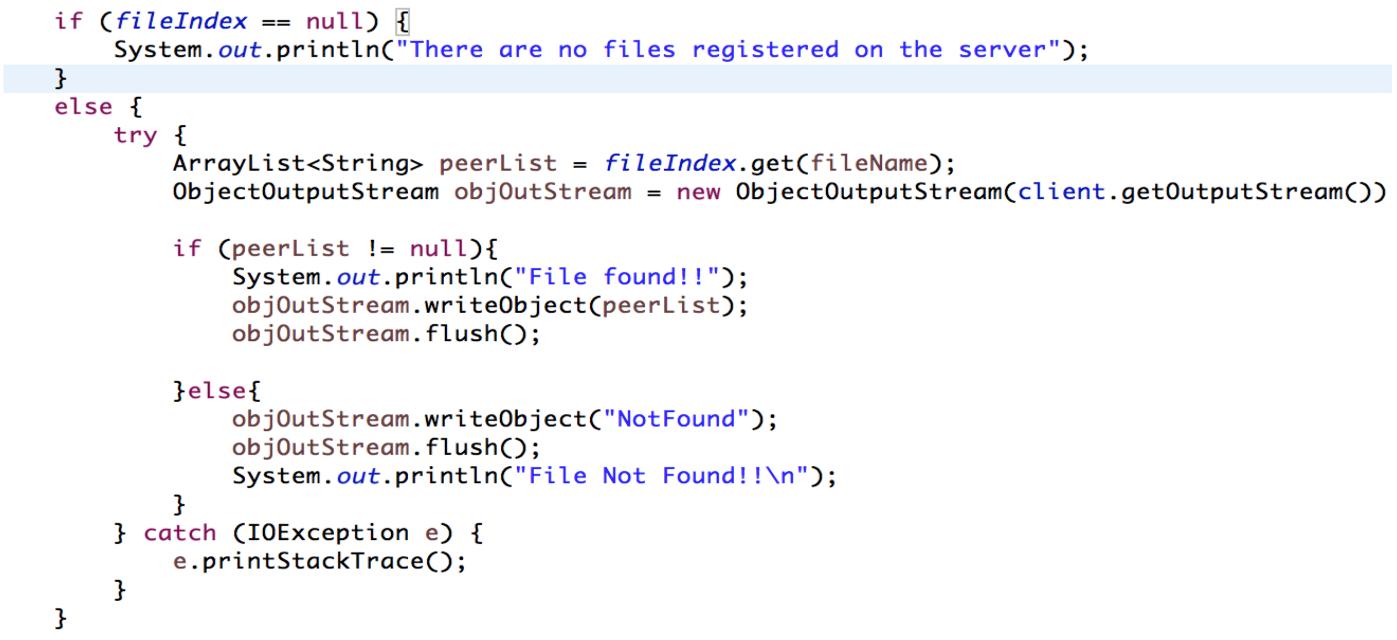
```
// Start server socket to accept request from peer.
serverSocket = new ServerSocket(port);
System.out.println("Server Started\n");

while (true){
    Socket client = serverSocket.accept();
    // With threading approach, this can accept multiple requests from peer
    Thread clientThread = new Thread(new HandlePeers(client));
    clientThread.start();
}
```

3.1.2 Register



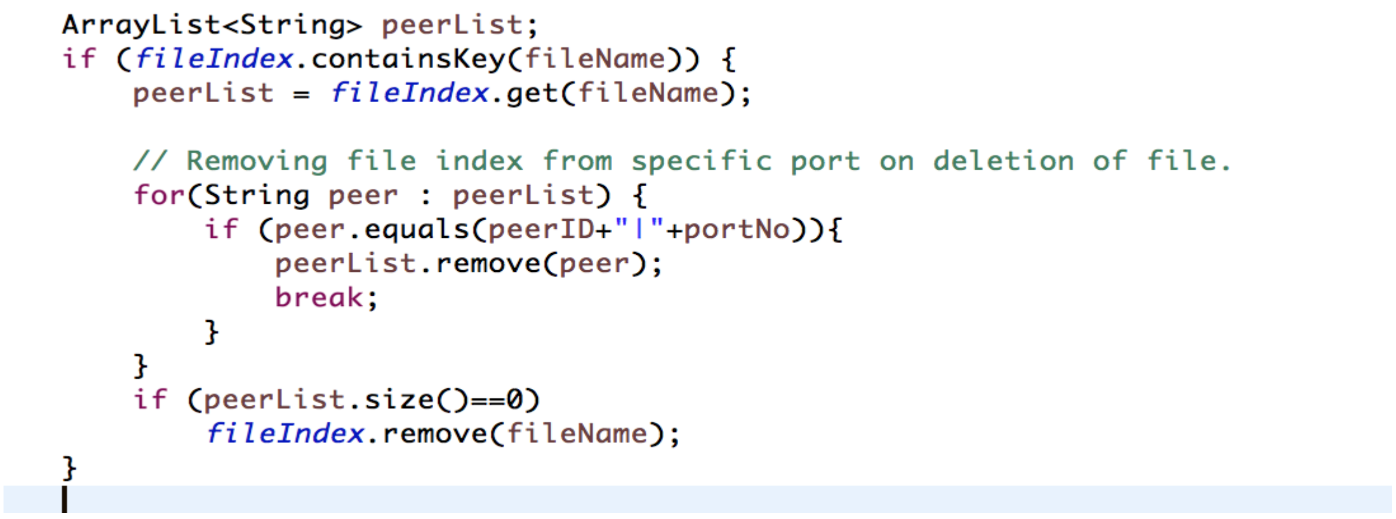
```
ObjectOutputStream objOutStream = new ObjectOutputStream(client.getOutputStream());

// Check if file already  registered on the server if yes then it add new peer to the list
// otherwise create new index
Boolean flag = true;
if (fileIndex.containsKey(fileName)) {
    peerList = fileIndex.get(fileName);

    for (Iterator<String> it = peerList.iterator(); it.hasNext(); ) {
        String peer = it.next();
        if (!peer.equalsIgnoreCase(peerID+"|"+portNo)) {
            flag = true;
        }
        else {
            flag = false;
            break;
        }
    }

    if (flag)
        peerList.add(peerID+"|"+portNo);

}
else {
    peerList = new ArrayList<String>();
    peerList.add(peerID+"|"+portNo);
    fileIndex.put(fileName, peerList);
}
objOutStream.writeObject(flag);
objOutStream.flush();
```

3.1.3 Search



```
if (fileIndex == null) {
    System.out.println("There are no files registered on the server");
}
else {
    try {
        ArrayList<String> peerList = fileIndex.get(fileName);
        ObjectOutputStream objOutStream = new ObjectOutputStream(client.getOutputStream())

        if (peerList != null){
            System.out.println("File found!!");
            objOutStream.writeObject(peerList);
            objOutStream.flush();

        }else{
            objOutStream.writeObject("NotFound");
            objOutStream.flush();
            System.out.println("File Not Found!!\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

3.1.4 Unregister

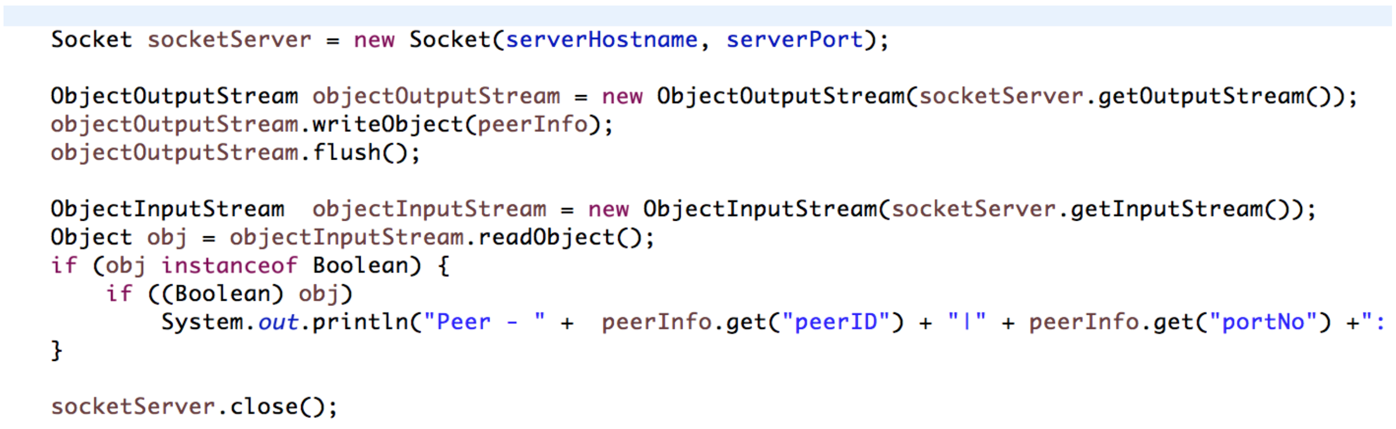


```
ArrayList<String> peerList;
if (fileIndex.containsKey(fileName)) {
    peerList = fileIndex.get(fileName);

    // Removing file index from specific port on deletion of file.
    for(String peer : peerList) {
        if (peer.equals(peerID+"|"+portNo)){
            peerList.remove(peer);
            break;
        }
    }
    if (peerList.size()==0)
        fileIndex.remove(fileName);
}
```

3.2 Client

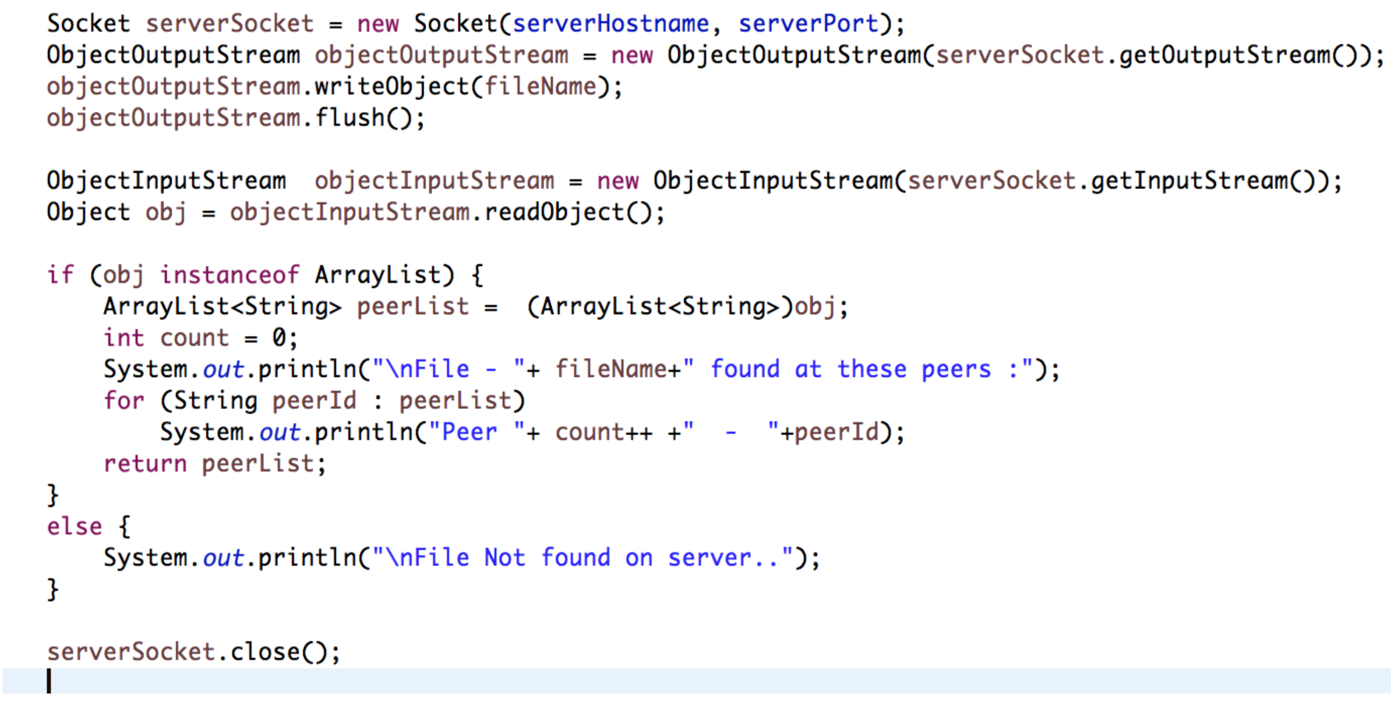
Peer Client accepts the data from user and performs various operations accordingly. In addition, it also prompts to user for additional inputs depending upon the operation selected. Peer Client uses Socket to communicate with Indexing Server and Peer Server.

3.2.1 Register



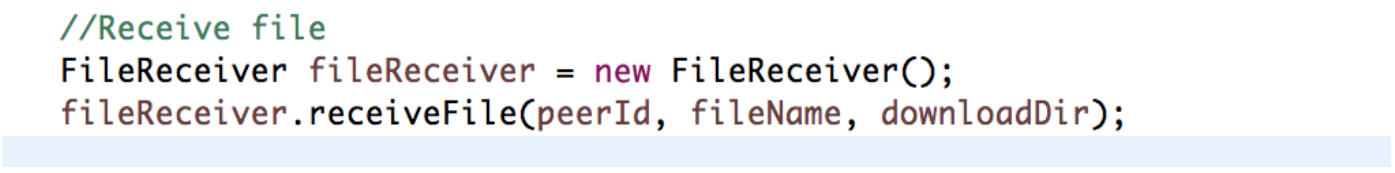
```
Socket socketServer = new Socket(serverHostname, serverPort);

ObjectOutputStream objectOutputStream = new ObjectOutputStream(socketServer.getOutputStream());
objectOutputStream.writeObject(peerInfo);
objectOutputStream.flush();

ObjectInputStream  objectInputStream = new ObjectInputStream(socketServer.getInputStream());
Object obj = objectInputStream.readObject();
if (obj instanceof Boolean) {
    if ((Boolean) obj)
        System.out.println("Peer - " +  peerInfo.get("peerID") + "|" + peerInfo.get("portNo") +":
}

socketServer.close();
```

3.2.2 Search



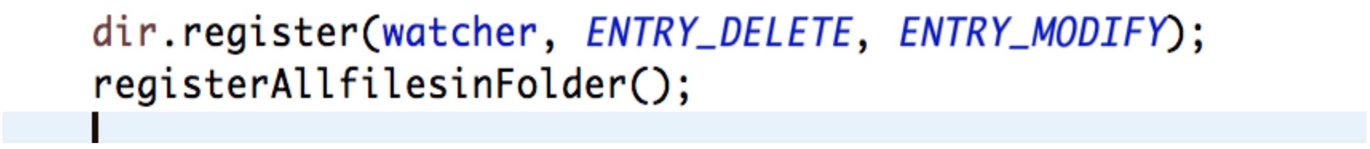
```
Socket serverSocket = new Socket(serverHostname, serverPort);
ObjectOutputStream objectOutputStream = new ObjectOutputStream(serverSocket.getOutputStream());
objectOutputStream.writeObject(fileName);
objectOutputStream.flush();

ObjectInputStream  objectInputStream = new ObjectInputStream(serverSocket.getInputStream());
Object obj = objectInputStream.readObject();

if (obj instanceof ArrayList) {
    ArrayList<String> peerList =  (ArrayList<String>)obj;
    int count = 0;
    System.out.println("\nFile - "+ fileName+" found at these peers :");
    for (String peerId : peerList)
        System.out.println("Peer "+ count++ +"  -  "+peerId);
    return peerList;
}
else {
    System.out.println("\nFile Not found on server..");
}

serverSocket.close();
```

3.2.3 Download



```
//Receive file
FileReceiver fileReceiver = new FileReceiver();
fileReceiver.receiveFile(peerId, fileName, downloadDir);
```

3.2.4 Auto indexing



```
dir.register(watcher, ENTRY_DELETE, ENTRY_MODIFY);
registerAllfilesinFolder();
```

4. Future Enhancements

- Once Indexing server shutdown, the files information which are registered gets erased, hence our future enhancement will be to save that information to text file or into database which can be used afterwards when server starts.

- We should pass full name of file which has to be searched/download, hence our future enhancement will be to apply search algorithms to find the partial matches.

5. Assumptions and Constraints

- If the peer which has file is closed, then the other peer which wants to download that file will not be able to connect and download the file.

- Peer should type exact file name that should be searched else the server will return message saying “File not found”.