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.

```
// connect to the index server on port 8101.
socket = new Socket("127.0.0.1", 8101);
int socketPort = socket.getLocalPort();
```

2. Send message:

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 hash map,

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

3. *Hash map operation.*

We use hash map to store peer and its file list.
Key is an integer class, and value is a string class.

```
localFileList = new HashMap<Integer, String>();
```

Hash map put and update,

```
globalFileList.put(tempPeer, tempList);
```

Get all elements in a hash map.

```
for (Map.Entry me : globalFileList.entrySet()) {
    System.out.println(me.getKey() + "    " + me.getValue());
}
```

4. *File Downloading operation.*

Read file content from source peer:

```
//  get the localfile location
String filePath = "/Users/yangkklt/cs550demo/"
        + Integer.toString(clientId - 2000) + "/"
        + downloadFileName;

File file = new File(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");
str = str.substring(0, textLength);
//  send the buffer to the target.
dos.writeUTF(str);
```

Write to a local file:

```
// write to local file
FileWriter fw = new FileWriter("/Users/yangkklt/cs550demo/"
        + Integer.toString(clientId) + "/" + fileString);

fw.write(result, 0, result.length());
fw.flush();
fw.close();
```

5. Multithread operation:

We use to implement runnable interface.

```
public class ClientClass implements Runnable {
    public int clientId;
```

We need to override the run() function:

```
//  this is the way to fork a thread. A new generated thread is responsible to
//  handle the download request.
@Override
public void run() {
```

Create a thread.

```
// create a new thread to set up a server socket
new Thread(new ClientClass(clientId + 2000)).start();
```

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

6. Automatic update (registry)

We use vfs2 to implement this function. We don't quite understand the meaning of this method, just know how to use it as a tool.

```java
//  this function uses vfs2 to monitor the change of a given directory.
//  It based on event-driven. file created, delete, change can trigger an event
//  that is synchronized void updateRegister. And I put the registry in this function
//  By doing this, each time an event is triggered, the client can do a registry.
public void initFileMonitor(String filePath) {
    try {
        FileSystemManager fsManager = VFS.getManager();
        FileObject listendir = fsManager.resolveFile(filePath);

        DefaultFileMonitor fm = new DefaultFileMonitor(new FileListener() {
            private synchronized void updateRegister() {
                getFileList("/Users/yangkklt/cs550demo/"
                        + Integer.toString(clientId));
                registry();
            }

            @Override
            public void fileCreated(FileChangeEvent fce) throws Exception {
                this.updateRegister();
            }

            @Override
            public void fileDeleted(FileChangeEvent fce) throws Exception {
                this.updateRegister();
            }

            @Override
            public void fileChanged(FileChangeEvent fce) throws Exception {
                this.updateRegister();
            }
        });
        fm.setRecursive(false);
        fm.addFile(listendir);
        fm.start();
    } catch (FileSystemException ex) {
        Logger.getLogger(ClientClass.class.getName()).log(Level.SEVERE,
                null, ex);
    }

}
```