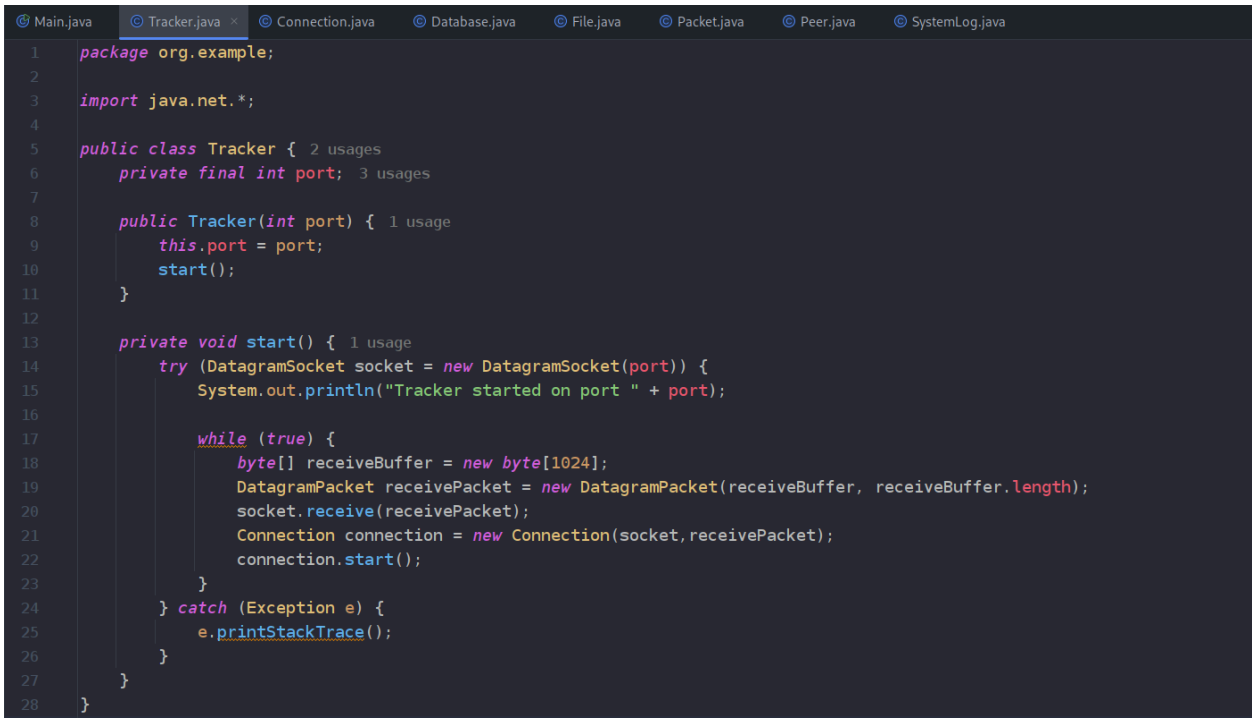
ملیکا علیزاده 401106255

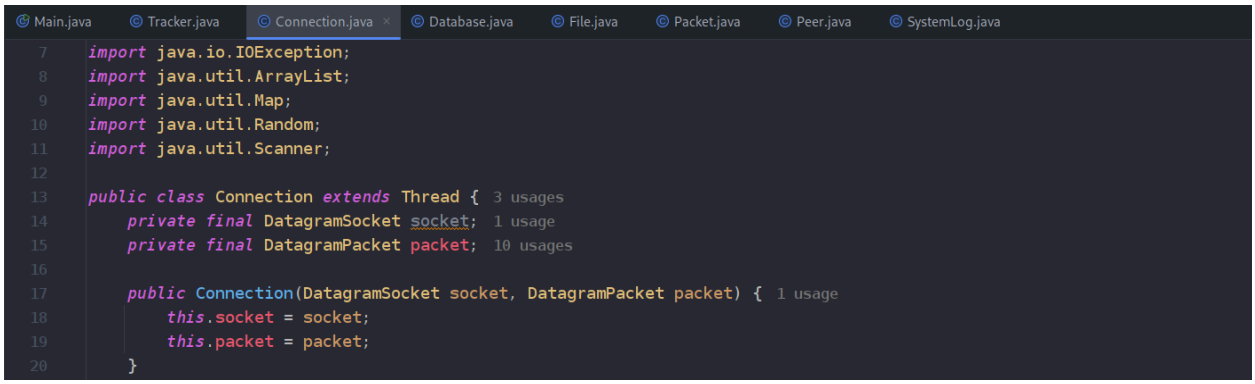
پیاده‌سازی Tracker:

ابتدا با شروع از تابع main یک tracker روی پورت 8080 میسازیم.

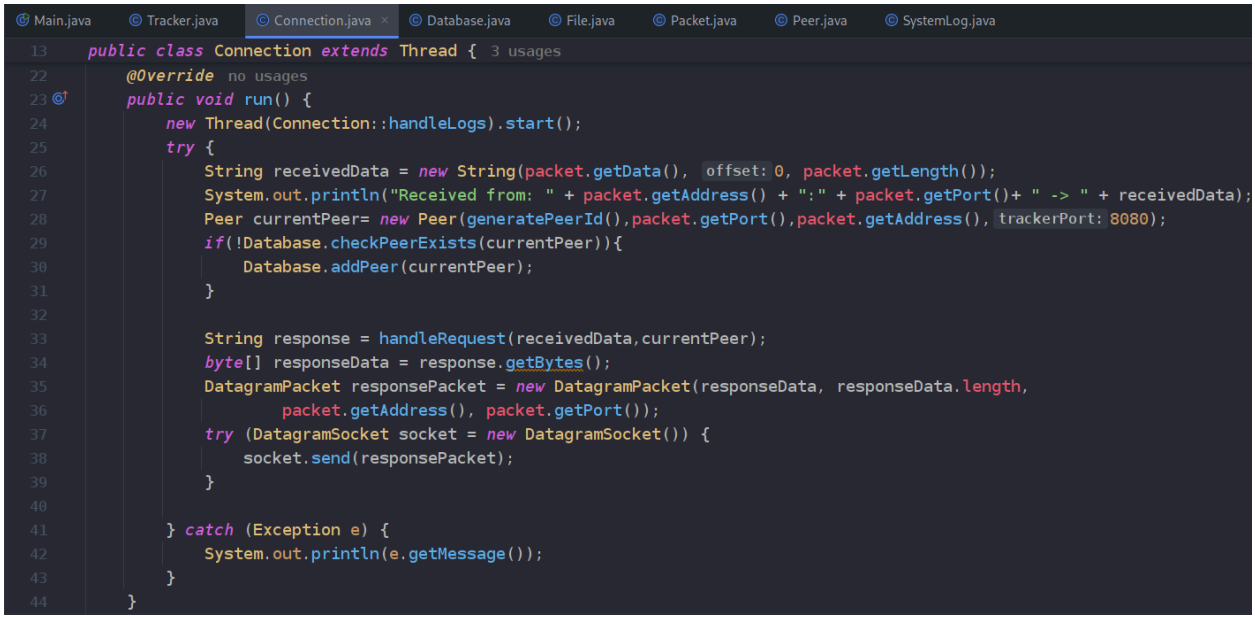
حال در tracker و در تابع start با استفاده از پورت داده شده و کلاس DatagramSocket یک کانکشن UDP ایجاد می‌کنیم. در آغاز کار tracker پورت آن را چاپ می‌کنیم. در یک حلقه بی‌نهایت پیام‌ها را از peers دریافت می‌کنیم به اینصورت که یک بافر برای ذخیره داده دریافتی درنظر می‌گیریم و با استفاده از کلاس DatagramPacket بسته را روی پورت موردنظر دریافت می‌کنیم. سپس در کلاس connection یک Thread برای پردازش درخواست ایجاد می‌شود تا برنامه به صورت multithread اجرا شود.



سپس در کلاس connection ابتدا مقادیر اولیه ورودی مقداردهی می‌شوند.



حال در تابع run ابتدا یک thread جدید برای پردازش دستورات لاگ ایجاد می‌شود. سپس داده‌های دریافتی از بسته UDP خوانده و چاپ می‌شود. همچنین با استفاده از کلاس peer یک آبجکت جدید ایجاد کرده و لیست اطلاعات peerها آپدیت می‌شود. در نهایت داده دریافتی از peer در تابع handleRequest پردازش شده و پاسخ تولید شده به peer فرستنده ارسال می‌شود.



در کلاس peer اطلاعات زیر برای هر peer در هنگام ایجاد ذخیره می‌شود. همچنین لیستی برای فایل‌های به اشتراک گذاشته شده توسط هر peer نیز در نظر گرفته شده است.

Main.java | Tracker.java | Connection.java | Database.java | File.java | Packet.java | Peer.java | SystemLog.java

```java
public class Peer {  18 usages
    private String id;  3 usages
    private int port;  3 usages
    private InetAddress ip;  3 usages
    private int trackerPort;  3 usages
    private ArrayList<File> files;  3 usages

    public Peer(String id, int port, InetAddress ip, int trackerPort) {  1 usage
        this.id = id;
        this.port = port;
        this.ip = ip;
        this.trackerPort = trackerPort;
        this.files = new ArrayList<>();
    }
```

در کلاس فایل دو ویژگی اسم و پورت مربوط به آن درنظر گرفته می‌شود.

Main.java | Tracker.java | Connection.java | Database.java | File.java | Packet.java | Peer.java | SystemLog.java

```java
package org.example;

public class File {  8 usages
    private String name;  3 usages
    private int port;  3 usages

    public File(String name, int port) {  3 usages
        this.name = name;
        this.port = port;
    }
```

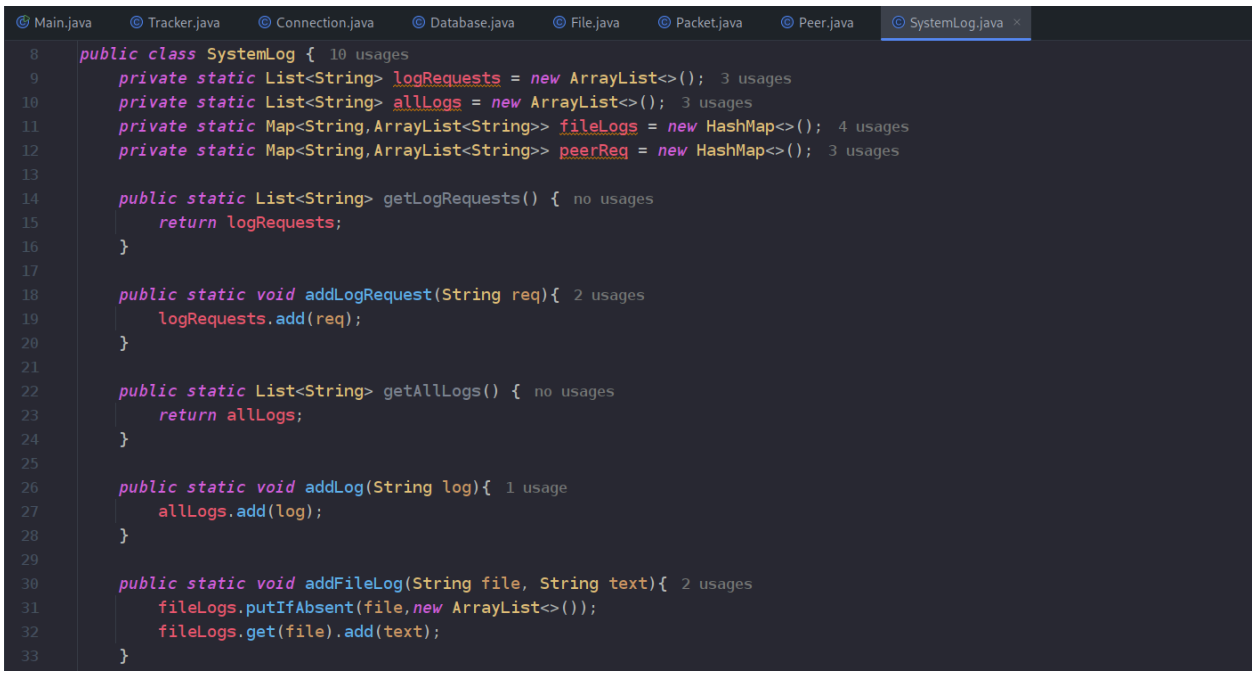
در کلاس Database دو لیست foundPeers و allPeers داریم که allPeers لیست تمام peerهای متصل است و foundPeers برای نمایش لاگ‌ها استفاده می‌شود.

Main.java | Tracker.java | Connection.java | Database.java | File.java | Packet.java | Peer.java | SystemLog.java

```java
public class Database {  7 usages
    public static ArrayList<Peer> foundPeers = new ArrayList<>();  3 usages
    private static ArrayList<Peer> allPeers = new ArrayList<>();  7 usages


    public static void addPeer(Peer peer) {  1 usage
        allPeers.add(peer);
    }

    public static void removePeer(Peer peer) {  1 usage
        for (Peer p : allPeers) {
            if (p.getPort() == peer.getPort()) {
                allPeers.remove(peer);
                return;
            }
        }
    }

    public static boolean checkPeerExists(Peer peer) {  1 usage
        for (Peer p : allPeers) {
            if (p.getPort() == peer.getPort()) {
                return true;
            }
        }
        return false;
    }
```

در تابع handleLogs دستورات لاگ در خط فرمان tracker را مدیریت می‌کند و به ترتیب هر کدام برای نمایش درخواست‌های get و موفقیت یا عدم موفقیت آن، درخواست‌های share و وضعیت ارسال و دریافت یک فایل مشخص استفاده می‌شود.

کلاس SystemLog برای مدیریت و ذخیره لاگ‌ها استفاده می‌شود. در آن لاگ درخواست‌ها و تمام لاگ‌ها و لاگ‌های مربوط به هر فایل و درخواست‌های هر peer به صورت لیست‌ها و map ذخیره می‌شود.



```java
public class SystemLog { 10 usages
    public static void addPeerReq(String file, String text){ 1 usage
        peerReq.putIfAbsent(file,new ArrayList<>());
        peerReq.get(file).add(text);
    }

    public static void printRequests(){ 1 usage
        logRequests.forEach(System.out::println);
    }

    public static void printLogs(){ 1 usage
        allLogs.forEach(System.out::println);
    }

    public static void printFileLogs(String fileName){ 1 usage
        if (fileLogs.containsKey(fileName)) {
            System.out.println("Logs for file " + fileName + ": ");
            fileLogs.get(fileName).forEach(System.out::println);
        } else {
            System.out.println("Error: File not found.");
        }
    }

    public static Map<String, ArrayList<String>> getPeerReq() { 1 usage
        return peerReq;
    }
}
```

در تابع handleRequest درخواست‌های دریافتی از peer پردازش می‌شود. سپس برای راحتی پردازش درخواست با استفاده از فرمت کلاس packet تجزیه می‌شود.

```java
public class Connection extends Thread { 3 usages
    private String handleRequest(String data, Peer currentPeer) throws IOException { 1 usage
        if (data.isEmpty())
            return null;

        try {
            Packet packet = new Gson().fromJson(data, Packet.class);
            String topic = packet.topic;
            String value = packet.value;
            String command = packet.command;
            String extra = packet.extra;
```

در دستور exit باید peer از دیتابیس حذف شود و پیام قطع ارتباط چاپ شود و در دستور request logs لیستی از تمام درخواست‌های ثبت شده توسط peer چاپ می‌شود.

```java
public class Connection extends Thread { 3 usages
    private String handleRequest(String data, Peer currentPeer) throws IOException { 1 usage
            switch (command) {
                case "exit" ->{
                    Database.removePeer(currentPeer);
                    System.out.println("Peer" +" Disconnected!");
                    return ("Peer Disconnected");
                }
                case "request logs" -> {
                    StringBuilder sb = new StringBuilder();
                    for (Map.Entry<String, ArrayList<String>> entry : SystemLog.getPeerReq().entrySet()) {
                        sb.append("File: ").append(entry.getKey()).append("\n");
                        for (String request : entry.getValue()) {
                            sb.append("  → Request: ").append(formatRequest(request)).append("\n");
                        }
                    }
                    return sb.toString();
                }
```

در دستور get اگر فایل مورد نظر از قبل موجود بود تمام peerهایی که آن فایل را دارند پیدا کرده و سپس پورت یکی از آنها را به صورت تصادفی به peer ارسال می‌کند. اگر فایل موجود نباشد پیغام خطا را چاپ می‌کند. همچنین وضعیت لاگ‌ها را نیز آپدیت کرده و فایل را به لیست فایل‌هایی که peer دارد اضافه می‌کند.

```
Main.java  Tracker.java  Connection.java  Database.java  File.java  Packet.java  Peer.java  SystemLog.java
13  public class Connection extends Thread { 3 usages
62      private String handleRequest(String data, Peer currentPeer) throws IOException { 1 usage
89                  case "get" -> {
90                      ArrayList<Peer> foundPeers = Database.getFilePort(extra);
91                      if (!foundPeers.isEmpty()) {
92                          SystemLog.addFileLog(extra, text: currentPeer.getId() + ": " + command);
93                          SystemLog.addLogRequest(currentPeer.getId() + ": " + command + " " + extra + " "
94                                  + ", Successful" + "\n" + foundPeersToString(foundPeers));
95                          Random random = new Random();
96                          Peer randomPeer = foundPeers.get(random.nextInt(foundPeers.size()));
97                          SystemLog.addPeerReq(extra, text: currentPeer.getId()+ "get"+extra+"from"+randomPeer.getId());
98                          int filePort=Database.findFilePort(randomPeer.getPort(),extra);
99                          currentPeer.addFile(new File(extra,Integer.parseInt(topic)));
100                         Database.updatePeer(topic,extra);
101                         return String.valueOf(filePort);
102                     } else {
103                         SystemLog.addLogRequest(generatePeerId() + ": " + command + extra +
104                                 ",Fail" + "\n No Peer found for this file");
105                         return "400: No such file exists";
106                     }
107                 }
```

در دستور share اطلاعات فایل به دیتابیس اضافه شده و به لیست فایل‌های peer و لاگ‌ها نیز اضافه می‌شود.

```
Main.java  Tracker.java  Connection.java  Database.java  File.java  Packet.java  Peer.java  SystemLog.java
13  public class Connection extends Thread { 3 usages
62      private String handleRequest(String data, Peer currentPeer) throws IOException { 1 usage
108                 case "share" -> {
109                     SystemLog.addFileLog(extra, text: currentPeer.getId() + " shared");
110                     SystemLog.addLog(currentPeer.getId() + " shared " + extra);
111                     currentPeer.addFile(new File(extra,Integer.parseInt(topic)));
112                     Database.updatePeer(topic,extra);
113                     return "200: "+extra+" shared successfully!";
114                 }
115                 default -> {
116                     return ("400: Invalid command.");
117                 }
118             }
119         } catch (JsonSyntaxException e) {
120             return ("400: Missing topic, value, or command fields.");
121         }
122     }
```

در آخر نیز اگر دستور ناشناخته باشد پیغام خطا چاپ می‌شود.

دو تابع زیر نیز برای ساخت رشته‌ای که قرار است چاپ شود است.

```
124 @   private static String formatRequest(String request) { 1 usage
125         return request.replaceAll( regex: "(peer\\d+)(get)(.+)(from)(peer\\d+)", replacement: "$1 $2 $3 $4 $5");
126     }
127
128 @   private String foundPeersToString(ArrayList<Peer> foundPeers) { 1 usage
129         StringBuilder sb = new StringBuilder();
130         for(Peer p:foundPeers){
131             sb.append(p.getId()).append(", ");
132         }
133         return sb.toString();
134     }
```

در کلاس packet درخواست به 4 بخش دستور، پورتی که در آن گوش می‌دهد، پورت tracker و فایل مورد نظر تقسیم می‌شود.

Main.java | Tracker.java | Connection.java | Database.java | File.java | Packet.java | Peer.java | SystemLog.java

```java
package org.example;

import ...

public class Packet implements Serializable { 2 usages
    String command; 2 usages
    String topic; 2 usages
    String value; 2 usages
    String extra; 2 usages

    public Packet(String command, String topic, String value, String extra) { no usages
        this.command = command;
        this.topic = topic;
        this.value = value;
        this.extra = extra;
    }

    public String toJson() { return new Gson().toJson( src: this); }
}
```

پیاده‌سازی Peer:

ابتدا با شروع از تابع main یک peer روی پورت 8080 میسازیم.

Main.java | Peer.java | Packet.java

```java
package org.example;

import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        Peer peer = new Peer( host: "localhost", port: 8080);
    }
}
```

در کلاس peer ابتدا آدرس و پورت و دیگر متغیرهای ثابت کلاس را مقداردهی می‌کنیم.

Main.java | Peer.java | Packet.java

```java
public class Peer { 3 usages
    private static final String TRACKER_ADDRESS = "127.0.0.1"; 4 usages
    private static final int TRACKER_PORT = 8080; 4 usages
    private static int PEER_SERVER_PORT; 4 usages
    private static Set<String> sharedFiles = new HashSet<>(); 2 usages
    private String serverAddress; 2 usages
    private int serverPort; 2 usages

    public Peer(String host, int port) throws IOException { 1 usage
        this.serverAddress = host;
        this.serverPort = port;
```

در این بخش ابتدا در یک thread جداگانه یک سرور فایل و یک سوکت برای ارتباط UDP با tracker ایجاد می‌کنیم و بعد حلقه‌ای برای گرفتن دستورات از کاربر ایجاد می‌شود.

در تابع startFileServer یک سرور فایل محلی روی پورت آزاد بین دو peer ایجاد می‌شود و هر بار که یک درخواست فایل به سرور می‌رسد، یک رشته جدید برای پردازش درخواست ایجاد می‌شود.

در تابع handleFileRequest پردازش درخواست‌های فایل از Peers انجام می‌شود. اگر فایل خواسته‌شده در لیست فایل‌های به اشتراک گذاشته‌شده باشد، فایل ارسال می‌شود. در غیر این صورت، یک پیام خطا ارسال می‌شود.

```java
public class Peer { 3 usages
    private static void handleFileRequest(Socket clientSocket) { 1 usage
        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
            OutputStream out = clientSocket.getOutputStream();

            String fileName = reader.readLine();
            System.out.println("Received request for file: " + fileName);


            if (sharedFiles.contains(fileName)) {
                System.out.println("File '" + fileName + "' sent to " + clientSocket.getInetAddress().getHostAddress());
            } else {
                out.write("ERROR: File not found\n".getBytes());
                System.out.println("Requested file '" + fileName + "' not found.");
            }

            clientSocket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

اگر دستور exit بود در تابع endPeer یک دستور برای tracker ارسال می‌شود و با چاپ پیغام آن برنامه تمام می‌شود.

```java
public class Peer { 3 usages
    private static void endPeer(DatagramSocket socket) throws IOException { 1 usage
        String message = new Packet( command: "exit",  topic: "",  value: "",  extra: "").toJson();
        byte[] data = message.getBytes();
        DatagramPacket packet = new DatagramPacket(data, data.length,
                InetAddress.getByName(TRACKER_ADDRESS), TRACKER_PORT);
        socket.send(packet);
        System.out.println(receivePacket(socket));
    }
```

اگر دستور request logs بود در تابع requestLogs درخواست دریافت لاگ به tracker ارسال شده و پاسخ آن چاپ می‌شود.

```java
public class Peer { 3 usages
    private static void requestLogs(DatagramSocket socket) throws IOException { 1 usage
        String message = new Packet( command: "request logs",  topic: "",  value: "",  extra: "").toJson();
        byte[] data = message.getBytes();
        DatagramPacket packet = new DatagramPacket(data, data.length,
                InetAddress.getByName(TRACKER_ADDRESS), TRACKER_PORT);
        socket.send(packet);
        System.out.println(receivePacket(socket));
    }
```

اگر دستور share بود در تابع registerFile یک پیغام share به tracker ارسال شده تا فایل در لیست فایل‌ها ذخیره شود.

```java
public class Peer { 3 usages
    private static void registerFile(String fileName, DatagramSocket socket) throws IOException { 1 usage
        String message = new Packet( command: "share", String.valueOf(PEER_SERVER_PORT), value: "8080", fileName).toJson();
        byte[] data = message.getBytes();
        DatagramPacket packet = new DatagramPacket(data, data.length,
                InetAddress.getByName(TRACKER_ADDRESS), TRACKER_PORT);
        socket.send(packet);
        System.out.println(receivePacket(socket));
        System.out.println("File registered with tracker: " + fileName);
    }
```

اگر دستور get بود در تابع requestFile یک دستور دریافت فایل به tracker ارسال می‌شود اگر فایل موجود باشد پورت peer را از tracker دریافت می‌کند.

```java
public class Peer { 3 usages
    private static void requestFile(String fileName, DatagramSocket socket) throws IOException { 1 usage
        String message = new Packet( command: "get", String.valueOf(PEER_SERVER_PORT), value: "8080", fileName).toJson();
        byte[] data = message.getBytes();
        DatagramPacket packet = new DatagramPacket(data, data.length,
                InetAddress.getByName(TRACKER_ADDRESS), TRACKER_PORT);
        socket.send(packet);

        byte[] buffer = new byte[1024];
        DatagramPacket responsePacket = new DatagramPacket(buffer, buffer.length);
        socket.receive(responsePacket);
        String response = new String(responsePacket.getData(), offset: 0, responsePacket.getLength());

        if (response.equals("400: No such file exists")) {
            System.out.println("No peers have this file.");
        } else {
            System.out.println("Peers with file: " + response);
            String peerAddress = "127.0.0.1";
            int peerPort = Integer.parseInt(response);
            downloadFile(peerAddress, peerPort, fileName);
        }
    }
```

سپس در تابع downloadFile فایل را از peer دیگر دریافت کرده و آن را ذخیره می‌کند.

```java
public class Peer { 3 usages
    private static void downloadFile(String peerAddress, int peerPort, String fileName) { 1 usage
        try {
            Socket socket = new Socket(peerAddress, peerPort);
            OutputStream out = socket.getOutputStream();
            out.write((fileName + "\n").getBytes());

            FileOutputStream fos = new FileOutputStream( name: "downloaded_" + fileName);
            InputStream in = socket.getInputStream();
            byte[] buffer = new byte[1024];
            int bytesRead;

            while ((bytesRead = in.read(buffer)) != -1) {
                fos.write(buffer, off: 0, bytesRead);
            }

            fos.close();
            in.close();
            socket.close();
            System.out.println("File '" + fileName + "' downloaded from " + peerAddress + ":" + peerPort);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

دو تابع کمکی زیر برای ارسال و دریافت پیغام‌ها به tracker استفاده می‌شوند.

```java
    private void sendPacket(DatagramSocket clientSocket, String m) throws IOException {  no usages
        String message = m;
        byte[] sendData = message.getBytes();
        InetAddress serverIP = InetAddress.getByName(serverAddress);
        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverIP, serverPort);
        clientSocket.send(sendPacket);
        System.out.println("Sent: " + message);
    }

    private static String receivePacket(DatagramSocket clientSocket) throws IOException {  3 usages
        byte[] receiveBuffer = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
        clientSocket.receive(receivePacket);
        String receivedData = new String(receivePacket.getData(),  offset: 0, receivePacket.getLength());
        return receivedData;
    }
}
```

اجرای برنامه:

ابتدا برنامه tracker را اجرا می‌کنیم.

```
Run   Run ×
/usr/lib/jvm/java-21-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar=40773:/usr/share/idea/bin -Dfile.encoding=
Tracker started on port 8080
```

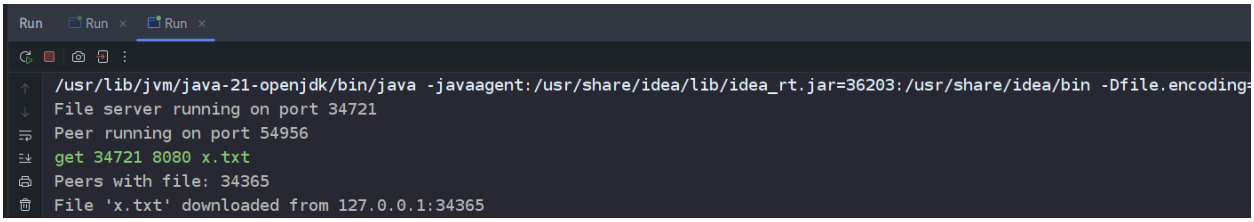
سپس برای مثال دو برنامه peer را ران می‌کنیم.

```
Run   Run ×   Run ×
/usr/lib/jvm/java-21-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar=44177:/usr/share/idea/bin -Dfile.encoding=
File server running on port 34365
Peer running on port 40964
```

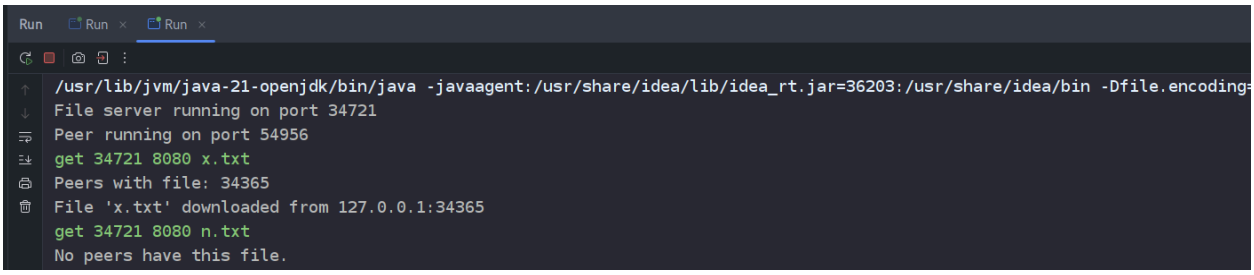
```
Run   Run ×   Run ×
/usr/lib/jvm/java-21-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar=36203:/usr/share/idea/bin -Dfile.encoding=
File server running on port 34721
Peer running on port 54956
```

حال برای مثال در یک peer یک فایل را share می‌کنیم و در دیگری آن فایل را دریافت می‌کنیم.

```
Run   Run ×   Run ×
/usr/lib/jvm/java-21-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar=44177:/usr/share/idea/bin -Dfile.encoding=
File server running on port 34365
Peer running on port 40964
share 34365 8080 x.txt
200: x.txt shared successfully!
File registered with tracker: x.txt
Received request for file: x.txt
File 'x.txt' sent to 127.0.0.1
```



اگر درخواست برای فایلی بدهیم که وجود ندارد:



حال اگر دستورات لاگ مربوط به tracker را اجرا کنیم:

و درخواست گرفتن لاگ در peer: