1. Các thành phần có trong Module P2P:
2. Package me.specify:

* Package này chứa các interface đặc tả toàn bộ các thành phần và chức năng cốt lõi từng thành phần cần phải có trong module P2P.
* Những interface của package:
  + 1. Interface P2P Protocol - IP2PProtocol: đặc tả những hằng số và những method các node trong mạng cần phải implement, bao gồm cả Bootstrap Node và Peer Node:
  + SC\_TIME\_OUT = 30000: thời gian từ khi một kết nối phát sinh đến khi hết hạn không kết nối được là 30 giây.
  + BACK\_LOG = 10: số lượng kết nối tối đa có thể kết nối đến một Socket Server đang lắng nghe là 10 kết nối.
  + **public** **void** listenRequest(): khởi động node lắng nghe các message từ các node khác trên Socket Server đã được chỉ định và tạo ra. Ở đây Peer Node sẽ lắng nghe message từ các node khác ở port 6868 và Bootstrap Node sẽ lắng nghe message ở port 8686.
  + **public** **void** shutdown(): kết thúc hoạt động của một node.
  + **public** **void** handleLeaveRequest(JSONObject requestPeerInfo): xử lý yêu cầu rời mạng từ node gửi yêu cầu rời mạng.
  + **public** **void** handleUpdateRequest(JSONObject requestPeerInfo): xứ lý yêu cầu cập nhật thông tin của một node đã thay đổi thông tin về username hoặc address.
    1. Interface IBootstrap – kế thừa interface IP2PProtocol: đặc tả những method cần thiết của IBootstrap:
  + **public** **void** handleJoinMsg(JSONObject data): xử lý message JOIN từ Peer Node. Message JOIN này ở dạng JSON được đặc tả rõ trong gói message.
    1. Interface IPeer – kế thừa interface IP2PProtocol: đặc tả những method cần thiết của Peer Node.
  + **public** **void** joinRequest(): gửi request mang message JOIN đến Bootstrap Node.
  + **public** **void** leaveRequest(): gửi request mang message LEAVE đến những node khác.
  + **public** **void** updateRequest(): gửi request mang message UPDATE đến những node khác.
  + **public** **void** handleTransferRequest(JSONObject bstrListPeerInfo): xử lý respone mang message TRANSFER\_LIST từ Bootstrap Node sau khi Peer Node gửi message JOIN đến.
  + **public** **void** handleAddNodeRequest(JSONObject requestPeerInfo): xử lý request mang message ADD\_NODE từ node muốn những node khác thêm thông tin của mình vào.
    1. Interface MessageCallback: interface này được dùng để thông báo các sự kiện trong quá trình giao tiếp giữa hai node, bao gồm các sự kiện: bắt đầu gửi message, kết thúc gửi message, nội dung message trong phiên giao tiếp giữa hai node.
  + **public** **void** onMessageStart(): thông báo quá trình gửi message giữa hai node bắt đầu.
  + **public** **void** onMessageEnd(): thông báo quá trình gửi message giữa hai node kết thúc.
  + **public** **void** onMessage(JSONObject message): thông báo nội dung message được gửi đến đối tượng nhận sự kiện xử lý message.
    1. Interface PeerCallback: interface này được dùng để thông báo các sự kiện cần quan tâm của Peer Node như:
  + **public** **void** onJoined(Peer peer): thông báo khi Peer Node đã nhận được danh sách Peer Node từ Bootstrap Node và đã trở thành một node trong mạng lưới P2P.
  + **public** **void** onAddedNode(PeerInfo peerInfo): thông báo khi Peer Node đã nhận và xử lý xong message ADD\_NODE từ node khác.
  + **public** **void** onUpdated(PeerInfo peerInfo): thông báo khi Peer Node đã nhận và xử lý xong message UPDATE từ node khác.
  + **public** **void** onLeaved(PeerInfo peerInfo): thông báo khi Peer Node đã nhận và xử lý xong message LEAVE từ node khác.
    1. Interface Data – IData: đặc tả những chức năng cần phải có của đối tượng quản lý dữ liệu. Vì mỗi một node đều thao tác, đọc, lưu thông tin của node trong file list\_peer.json và trạng thái của node trong status.json nên mỗi node đều cần có một đối tượng quản lý dữ liệu. Những method cần thiết cho một đối tượng quản lý dữ liệu:
  + **public** ArrayList<PeerInfo> getListPeerInfo(): lấy danh sách thông tin của các node có trong mạng, được gửi về từ Bootstrap Node sau khi Peer Node gửi message JOIN.
  + **public** **void** add(PeerInfo peerInfo): thêm thông tin một node vào trong danh sách node có sẵn.
  + **public** **void** remove(PeerInfo peerInfo): xóa thông tin một node trong danh sách.
  + **public** **void** update(PeerInfo peerInfo): cập nhật thông tin một node có sẵn trong danh sách.
  + **public** **boolean** isExist(PeerInfo peerInfo): kiểm tra sự tồn tại của một node với thông tin được cung cấp.

1. Package me.message:

* Các node trong mạng giao tiếp thông qua message được gửi bởi Request và Respone. Message được chuyển đi là dạng text có cấu trúc, sử dụng định dạng JSON. Mỗi một message là một đối tượng JSON bao gồm hai thành phần:
  + Message Type: loại thông điệp giao tiếp, được chỉ định bởi thuộc tính: “msg\_type”. Loại thông điệp chỉ định tác vụ cần được xử lý từ request được gửi. Ví dụ như message type JOIN tương đương với tác vụ tham gia mạng được gửi từ Peer Node đến Bootstrap Node.
  + Message Data: dữ liệu của thông điệp giao tiếp, được chỉ định bởi đối tượng: “msg\_data”. Đối tượng msg\_data chứa mảng JSON “list\_peeer”, trong mảng này chính là thông tin node cần giao tiếp.
* Để thực hiện đầy đủ thì package này có những class sau:
  + Class MessageJSONAttribute: chứa các thông điệp giao tiếp giữa các node ở dạng text, đồng thời cũng là JSON key được dùng để phân tích message ở dạng JSON trong quá trình giao tiếp. Class này có những thành phần sau:
    - *MSG\_TYPE* = "msg\_type": key JSON chỉ định loại message đang được giao tiếp. Được dùng để phân tích message thành đối tượng Message dùng trong chương trình từ message giao tiếp dưới dạng JSON.
    - *MSG\_DATA* = "msg\_data": key JSON chỉ định phần dữ liệu được mang theo bởi message. Được dùng để phân tích message thành đối tượng Message dùng trong chương trình từ message giao tiếp dưới dạng JSON.
    - *MSG\_TYPE\_JOIN* = "join": message JOIN, thông điệp này thông báo tới Bootstrap Node rằng Peer Node có yêu cầu tham gia mạng p2p, sau khi nhận được thông điệp này Bootstrap Node sẽ thêm thông tin Peer Node này vào **list\_peer** và gửi message ***TRANSFER\_LIST*** cùng với dữ liệu là danh sách **list\_peer** đến Peer Node vừa yêu cầu.
    - *MSG\_TYPE\_LEAVE* = "leave": message LEAVE, thông điệp này được gửi từ Peer Node đến Bootstrap Node thông báo rằng Peer Node muốn rời khởi mạng kèm theo dữ liệu là thông tin của Peer Node. Sau khi nhận được message này Bootstrap Node sẽ xóa thông tin Peer Node trong danh sách list\_peer và gửi message cập nhật đến tất cả các node trong mạng.
    - *MSG\_TYPE\_UPDATE* = "update": message UPDATE, thông điệp này được gửi từ Peer Node đến Bootstrap Node thông báo rằng Peer Node có sự thay đổi thông tin kèm theo thông tin mới. Bootstrap Node sẽ cập nhật lại thông tin của Peer Node yêu cầu và gửi message cập nhật đến cho tất cả các node trong mạng.
    - *MSG\_TYPE\_TRANSFERLIST* = "transfer\_list": message TRANSFER\_LIST, thông điệp này được gửi từ Bootstrap Node đến Peer Node. Sau khi nhận được message JOIN của Peer Node, Bootstrap Node sẽ xử lý như đã diễn tả ở phần message JOIN. Xử lý xong message JOIN, Bootstrap Node sẽ gửi lại message TRANSFER\_LIST kèm theo dữ liệu là danh sách thông tin tất cả các node trong mạng đến Peer Node. Peer Node sẽ xử lý message này và có được **list\_peer,** lúc này Peer Node đã chính thức tham gia vào mạng.
    - *MSG\_TYPE\_ADD\_NODE* = "add\_node": message ADD\_NODE, thông điệp này được gửi từ Peer Node đến Bootstrap Node thông báo rằng Peer Node muốn Bootstrap Node gửi message đến tất cả những node trong mạng thêm Peer Node này vào.
  + Enum EMsgType: trình bày những loại thông điệp ở dạng enum, tương đương với những message giao tiếp giữa các node ở dạng text. Enum này được tạo ra nhằm mục đích dễ dàng xử lý các loại message hơn ở dạng text. Enum này bao gồm những thành phần sau:
    - Enum JOIN: mang ý nghĩa tương đương message JOIN.
    - Enum LEAVE: mang ý nghĩa tương đương message LEAVE.
    - Enum UPDAT: mang ý nghĩa tương đương message UPDATE.
    - Enum TRANSFER\_LIST: mang ý nghĩa tương đương message TRANSFER\_LIST.
    - Enum ADD\_NODE: mang ý nghĩa tương đương message ADD\_NODE.
  + Class Message: thực thi message được sử dụng trong chương trình. Class Message gồm những thành phần sau:
    - EmsgType: loại thông điệp ở dạng enum.
    - JSONObject: dữ liệu của thông điệp.
    - **public** Message(JSONObject jsMsg): method khởi tạo thông điệp từ dữ liệu message thô JSON.
    - **public** Message(EMsgType msgType, JSONObject msgData): method khởi tạo thông điệp từ enum loại message EMsgType và dữ liệu thông điệp dạng JSONObject.
    - **public** EMsgType getMsgType(): get enum loại thông điệp.
    - **public** JSONObject getMsgData(): get dữ liệu thông điệp ở dạng JSON.
    - **public** JSONObject toJsonObject(): method chuyển đổi đối tượng message sang dạng JSON.
  + Class MessageParser: lớp tiện ích dùng để phân tích message từ thô dạng JSON sang đối tượng Message.
    - **public** MessageParser(JSONObject data): hàm khởi tạo đối tượng từ dữ liệu message dạng thô JSON.
    - **public** EMsgType getMessageType(): get loại thông điệp ở dạng enum.
    - **public** JSONObject getMessageData(): get dữ liệu thông điệp ở dạng JSON.
  + Class MessageDataParser: lớp tiện ích dùng để phân tích dữ liệu của message, mục đích chủ yếu của lớp này là phân tích dữ liệu message ở dạng JSON sang danh sách thông tin Peer.
    - **public** MessageDataParser(JSONObject msgData): hàm khởi tạo đối tượng từ dữ liệu message ở dạng JSON.
    - **public** ArrayList<PeerInfo> getListPeerInfo(): lấy danh sách thông tin peer được phân từ ra từ dữ liệu dạng JSON.

1. Package me.request:

* Một request là một đối tượng mang thông tin tác vụ được gửi trong phiên giao tiếp từ client node đến server node thông báo cho server node biết yêu cầu từ client node. Có các loại request như sau:
  + Request *START\_MSG* = "start\_msg": khi client node đã kết nối được với server node thì request này thông báo cho server node biết rằng client node muốn bắt đầu gửi message.
  + Sau khi client node thông báo cho server node bắt đầu quá trình gửi message thì client build một request mang nội dung message cần giao tiếp và gửi thông qua request.
  + Request *END\_MSG* = "end\_msg": sau khi gửi xong message thì client node gửi request này thông báo cho server node rằng client node đã hoàn tất gửi message và kết thúc phiên giao tiếp.

1. Package me.respone:
2. Package me.constant:

* Pakage này chứa interface PeerPort đặc tả port được sử dụng để giao tiếp giữa giữa Peer Node trong mạng với Bootstrap Node và giữa các Peer Node với nhau.
* Các port được sử dụng để giao tiếp trong thư viện này bao gồm:
  + PORT\_BOOTSTRAP (port 6868): port được Bootstrap Node dùng để lắng nghe các request từ Peer Node. Mỗi khi Peer Nodephát sinh một request đến Bootstrap Node ở port này, Bootstrap Node sẽ chấp nhận và tạo ra một Request Handler để xử lý các yêu cầu của Peer Node.
  + PORT\_PEER: port được Peer Node dùng để lắng nghe respone từ Bootstrap Node và request từ các Peer Node khác. Khi Peer Node này request yêu cầu lên Bootstrap Node, trong quá trình xử lý Bootstrap Node sẽ gửi respone về Peer Node thông qua cổng này. Hoặc khi các Peer Node khác phát sinh các request qua cổng này thì Peer Node sẽ chấp nhận và phát sinh một Request Handler để xử lý các messeage.

1. Package me.p2p:
2. Package me.log:
3. Package me.data: