# Program Listing

---

## Peer.java

```
/*------------------start change---------------------*/

	//The Map messageToPeer record the key of message ID with value of its hit peers ID for future obtaining and its states such as modify date, consistency state.
	private static ConcurrentHashMap<String, ConcurrentLinkedQueue<String[]>> msgHit = new ConcurrentHashMap<>();
	//The Queue maintain the sequence of message hit. when its size is hitsize and new hit is coming, the oldest one will be removed.
	private static ConcurrentLinkedQueue<String> hitlist= new ConcurrentLinkedQueue<String>();
	private static int hitsize = 20;

	//The Map originFile record the key of master copy filename with value of the file's info, such as version Number, modify date, consistency state.
	private static HashMap<String, FileInfo> originFile = new HashMap<>();
	//The Map cachedFile record the key of cached copy filename with value of the file's info, such as version Number, modify date, consistency state.
	private static HashMap<String, CachedInfo> cachedFile = new HashMap<>();

	//The Map messageToPeer record the key of invalidating message ID with value of its upstream peer ID. It's used for spread invalidating.
	private static ConcurrentHashMap<String, String> invalidmsgToPeer = new ConcurrentHashMap<>();
	//The Queue maintain the sequence of invalidating message. when its size is invalidmsgsize and new massage is coming, the oldest one will be removed.
	private static ConcurrentLinkedQueue<String> invalidmsglist= new ConcurrentLinkedQueue<String>();
	private static int invalidmsgsize = 20;

	//The default time-to-refresh. The dimension is second.
	private static int TTR = 1*60;
	private static String cachedfilepath = "cachedfiles"; //The relative path where saves those cached files.
	private static String mode = "None"; //The on/off of push or pull mode. Default is both off.
/*------------------end change———————————*/

/*------------------start change---------------------*/
	try {
		//If this peer has the file, call queryhit() back to upstream peer.
		//Change: Depends on whether it's master or cached copy, input different boolean identifier.
		if(originFile.containsKey(filename)){
```

```
                        PeerI peer = (PeerI) Naming.lookup(upstream);
                        peer.queryhit(message, maxTTL-TTL+1, filename, PeerUrl, Port,
originFile.get(filename), true);
                }else if(cachedFile.containsKey(filename) && cachedFile.get(filename).state ==
FileState.VALID){//For cached file, the state should be valid.
                        PeerI peer = (PeerI) Naming.lookup(upstream);
                        peer.queryhit(message, maxTTL-TTL+1, filename, PeerUrl, Port,
cachedFile.get(filename).info, false);
                }
/*------------------end change———————————*/

/*------------------start change----------------------*/
                                //Change: Add more info when queryhit, such as last-modify-time.
                                String[] peerUrl = new String[3];
                                peerUrl[0] = peerIP + ":" + portnumber + "/Peer"; //First record
sender peer IP.
                                SimpleDateFormat df=new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss");
                                peerUrl[1] = df.format(info.time.getTime()); //Second record file
modification time.
                                peerUrl[2] = origin? "Origin" : "Cached copy"; //Third record
whether it is origin.
                                msgHit.get(messageID).add(peerUrl);
                                hitlist.add(messageID);
/*------------------end change———————————*/

/*------------------start change----------------------*/
        //Implement obtain method required by assignment.
        //Read the file which name is filename received from other peer, save it to buffer and
send the buffer to the callee.
        //Change: return more info about the file, which is recorded in object filepack.
        public FilePackage obtain(String filename) throws RemoteException{
                File file;
                if(originFile.containsKey(filename)){
                        file = new File(filepath + "/" + filename);
                }else if (cachedFile.containsKey(filename)){
                        file = new File(cachedfilepath + "/" + filename);
                }else return null;
                if(!file.exists()){
                        System.out.println("Obtain Error: the file " +  filename +"is incorrectly
appear in originFile or cachedFile map.");
                        return null;
                }
                byte buffer[] = new byte[(int) file.length()];
                try {
                        //Read in the file.
                        BufferedInputStream filein = new BufferedInputStream(new
FileInputStream(file));
                        filein.read(buffer, 0, buffer.length);
```

```
                        filein.close();

                        //Encapsulate file data with its states into filepack and return to the peer
client.
                        if(originFile.containsKey(filename)){
                                FileInfo info = originFile.get(filename);
                                FilePackage filepack = new FilePackage(info.versionNo, info.time,
Calendar.getInstance(), TTR, PeerUrl+":"+Port+"/Peer", FileState.VALID, buffer);
                                return filepack;
                        }else{
                                CachedInfo info = cachedFile.get(filename);
                                FilePackage filepack = new FilePackage(info.info.versionNo,
info.info.time, info.cachedTime, info.TTR, info.originIP, info.state, buffer);
                                return filepack;
                        }
                } catch (Exception e) {
                        System.out.println("File Retrieve Error: " + e.getMessage());
                        return null;
                }
        }

        //Implement new invalidation function.
        //When invalidating, the messageID contains messageID and also upstreamID
separated by "#".
        public void invalidation(String messageID, String originIP, String filename, int versionNo,
int TTL){
                String[] IDs = messageID.split("#");
                String message = IDs[0];
                String upstream = IDs[1];

                //If this invalidation has not come to this peer, start checking.
                if(!invalidmsgToPeer.containsKey(message)){
                        //Maintain the invalidmsglist and msgToPeer.
                        while(invalidmsglist.size() >= invalidmsgsize){
                                String oldmsg = invalidmsglist.poll();
                                invalidmsgToPeer.remove(oldmsg);
                        }
                        invalidmsglist.add(message);
                        invalidmsgToPeer.put(message, upstream);

                        //If this peer has kept this cached file, check its version Number.
                        if(cachedFile.containsKey(filename) && cachedFile.get(filename).state ==
FileState.VALID){//For cached file, the state should be valid.
                CachedInfo info = cachedFile.get(filename);
                                int thisVersion = info.info.versionNo;
                                if(thisVersion < versionNo){
                                        info.state = FileState.INVALID;
                                        System.out.println("\nInvalidation Notice: The file "+
filename +" is invalid now.\n");
```

```
                                }
        }

        //keep transmit the invalidation query to other neighbors peers.
        if(TTL != 1){
                for(String str: neighbours){
                        PeerI peer;
                        if(str.equals(upstream)) continue; //Skip the upstream peer.
                        try {
                                peer = (PeerI) Naming.lookup(str);
                                peer.invalidation(message+"#"+PeerUrl+":"+Port+"/Peer", originIP,
filename, versionNo, TTL-1);
                        } catch(Exception e) {
                                System.out.println("Peer Invalidation - transmit Error: " +
e.getMessage());
                                return;
                        }
                }
        }
                }

        }

        //The new pull function simply check whether the versionNo is the latest one.
        //If versionNo is same, return fresh TTR. Otherwise, return 0. Additionally, if server
cannot find file in list, return -1;
        public int pull(String filename, int versionNo){
                if(!originFile.containsKey(filename)) return -1;
                FileInfo info = originFile.get(filename);
                if(info.versionNo > versionNo) return 0;
                else return TTR;
        }

/*--------------------end change----------------------*/

/*--------------------start change----------------------*/
                                System.out.println("6.Set TTR (now is :" + TTR +")");
                                System.out.println("7.Set push/pull mode. (now: " + mode + ")");
                                System.out.println("8.Simply modify file (change last-modify-time
to now).");
                                System.out.println("9.Show master copy and cached copy on this
peer.");
/*--------------------end change----------------------*/

/*--------------------start change----------------------*/
                                case 6:
                                        System.out.println("Please enter the TTR time(s):");
                                        TTR = getSelection(in, 1, Integer.MAX_VALUE);
```

```
                                        System.out.println("The TTR time has been set as: " + TTR
+".\n");
                                        break;
                                case 7:
                                        System.out.println("Please select a option mode: 1.Push
2.Pull 3.None");
                                        int choice = getSelection(in, 1, 3);
                                        switch(choice){
                                        case 1:
                                                if(pull != null){
                                                        pull.interrupt();
                                                        pull = null;
                                                }
                                                mode = "Push";
                                                break;
                                        case 2:
                                                pull = new PullMode();
                                                pull.start();
                                                mode = "Pull";
                                                break;
                                        case 3:
                                                if(pull != null){
                                                        pull.interrupt();
                                                        pull = null;
                                                }
                                                mode = "None";
                                        }
                                        break;
                                case 8:
                                        System.out.println("Please enter the file name of master
copy:");
                                        filename = in.nextLine();
                                        changeModifyTime(filename);
                                        break;
                                case 9:
                                        showCopyInfo(in);
                                        break;
/*-------------------end change---------------------*/

/*-------------------start change---------------------*/
                        //Initially record the master copy files information into map of originFile.
                        public void recordFile(){
        File path = new File(filepath);
        if(!path.exists()){
                System.out.println("Error: This path " + filepath +" is lost.");
                System.exit(-1);
        }
        //Record all the files under the path.
        for(String name: path.list()){
```

```
                File file = new File(filepath+"/"+name);
                Calendar cal = Calendar.getInstance();
                cal.setTimeInMillis(file.lastModified());//Get last modify time of this file.
                FileInfo info = new FileInfo(1, cal);
                originFile.put(name, info);
        }
                }

                //This thread will automatically update the record of master copy files. In push
mode, it will call invalidation if needed.
                public class Update extends Thread{
                        public void run() {
                                try {
                                        while(true){
                                                updateRecord();
                                                Thread.sleep(2000);//Check the file in every two
seconds.
                                        }
                                } catch (Exception e) {
                                        System.out.println("PushMode Exception: " +
e.getMessage());
                                return;
                                }
                        }
                }

                //This method will update the info for master copy files (in originFile).
                //In push mode, it will call invalidation if the last-modify-time of master copy was
changed.
                public void updateRecord(){
                        File path = new File(filepath);

        //Record all the files under the path.
        for(String name: path.list()){
                if(!originFile.containsKey(name)){//Ensure the originFile map is correct.
                        System.out.println("Warning: the master copy in /file does not exists in
map of originFile.");
                        continue;
                }
                File file = new File(filepath+"/"+name);
                Calendar cal = Calendar.getInstance();
                cal.setTimeInMillis(file.lastModified());//Get last modify time of this file.

                FileInfo info = originFile.get(name);
                //When the file in list is old, we should update the originFile and call invalidation
to neighbors.
                if(info.time.compareTo(cal)<0){
                        //Update map of originFile.
                        info.versionNo++;
```

```
                info.time = cal;

                if(mode.equals("Push")){
                        String messageID = peerid + "-" + messageNo; //Create a unique
message ID.
                                        messageNo++;
                                        String originIP = PeerUrl + ":" + Port + "/Peer";
                                        String querymsgID = messageID + "#" + originIP; //
Unique querymsgID combined with messageID and upstream Peer, uesd for query().
                        PeerI peer;
                        for(String str: neighbours){//Call invalidation to neighbors.
                                        try {
                                                peer = (PeerI) Naming.lookup(str);
                                                peer.invalidation(querymsgID,
originIP, name, info.versionNo, maxTTL);
                                        } catch(Exception e) {
                                                System.out.println("updateRecord -
transmit invalidation Error: " + e.getMessage());
                                                return;
                                        }
                                }
                }
        }
                }

                //This thread will automatically check/update the record of master copy files.
Used for pull mode.
                public class PullMode extends Thread{
                        public void run() {
                                try {
                                        int time = 0;
                                        while(true){
                                                for(CachedInfo e: cachedFile.values()){
                                                        if(e.state == FileState.VALID){//Check
whether expired
                                                                Calendar cal =
Calendar.getInstance();
                                                                cal.add(Calendar.SECOND, -
e.TTR);
                                                                if(e.cachedTime.compareTo(cal) < 0)
{//If last-modify-time exceed the range of TTR.
                                                                        e.state =
FileState.TTRexpired;
                                                                }
                                                        }
                                                }
                                                time++;
```

```
                                        if(time == 15){//For every 30s, call pull function if
the file state is TTR expired.
                                            for(Map.Entry<String, CachedInfo> entry:
cachedFile.entrySet()){
                                                if(entry.getValue().state ==
FileState.TTRexpired){
                                                    String filename =
entry.getKey();
                                                    CachedInfo info =
entry.getValue();
                                                    PeerI peer = (PeerI)
Naming.lookup(info.originIP);
                                                    int ans = peer.pull(filename,
info.info.versionNo);

                                                    if(ans <= 0){//Negative
means not valid.
                                                        info.state =
FileState.INVALID;
System.out.println("Pull Notice: the file " + filename + " is now invalid.");
                                                        if(ans == -1){
System.out.println("Pull Error: the file " + filename + "is not in Peer \"" + info.originIP + "\" ");
                                                        }
                                                    }else{
                                                        info.cachedTime =
Calendar.getInstance();
                                                        info.TTR = ans;
                                                        info.state =
FileState.VALID;//If still valid, Update the TTR.
                                                    }
                                                }
                                            }
                                            time = 0;
                                        }
                                        Thread.sleep(2000);//Check the file in every two
seconds.
                                    }
                                } catch (Exception e) {
                                    System.out.println("PullMode Exception: " +
e.getMessage());
                                    return;
                                }
                            }
                        }

                        public void changeModifyTime(String filename){
                            if(!originFile.containsKey(filename))
```

```
                                System.out.println("Warning: the origin File list does not contain
this file.");
                        File fileToChange = new File(filepath + '/' + filename);
                        if(!fileToChange.exists())
                                System.out.println("Error: the file does not exists but origin File list
contains it!");
                        if (fileToChange.setLastModified(System.currentTimeMillis())){
                                SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss");
                                System.out.println("Change last modify time to " +
df.format(fileToChange.lastModified()) +" Successfully!");
                        }
                        else
                                System.out.println("Error: Changing last modify time Failed!");
                }

                public void showCopyInfo(Scanner in){
                        System.out.println("----The information for master copy:");
                        if(originFile.isEmpty()) System.out.println("No master copy");
                        for(Map.Entry<String, FileInfo> e: originFile.entrySet()){
                                String filename = e.getKey();
                                FileInfo info = e.getValue();
                                SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss");
                                System.out.println(filename + "\t\tVersionNo: " + info.versionNo +
"\tLast-Modify-Time: " + df.format(info.time.getTime()));
                        }
                        System.out.println("----The information for cached copy:");
                        if(cachedFile.isEmpty()) System.out.println("No cached copy");
                        for(Map.Entry<String, CachedInfo> e: cachedFile.entrySet()){
                                String filename = e.getKey();
                                CachedInfo info = e.getValue();
                                SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss");
                                System.out.println(filename + "\t\tVersionNo: " +
info.info.versionNo + "\tLast-Modify-Time: " + df.format(info.info.time.getTime()) +" (" + info.state
+ ")" );
                                if(mode.equals("Pull"))
                                        System.out.println("\t\tTTR: " + info.TTR + "\tRefresh Time:
" + df.format(info.cachedTime.getTime()) );
                        }
                        System.out.println("Please press enter to return to the menu.");
                        in.nextLine();
                }

/*-------------------end change---------------------*/

/*-------------------start change---------------------*/
                        //Change: now the queue recored String[] object.
```

```
                        ConcurrentLinkedQueue<String[]> queue = msgHit.get(msgid);
                        if(queue == null){
                                System.out.println("No queried peer has this file "  + filename +
"." );
                                return;
                        }

                        String[][] peerids = new String[queue.size()][3];
                        int index = 0;
                        for(String[] e: queue){//Change: Extract all infos from queue.
                                peerids[index][0] = e[0];
                                peerids[index][1] = e[1];
                                peerids[index][2] = e[2];
                                index++;
                        }

                        if(queue.size() == 0) {
                                System.out.println("No queried peer has this file "  + filename +
"." );
                                return;
                        }

                        //Show all the peers who keep the file.
                        //Change: also show its last-modify-time and valid state.
                        System.out.println("There are all address of the peers keeping file " +
filename + ":");
                        for(int i=0; i<peerids.length; i++){
                                System.out.println((i+1) + ". " + peerids[i][0] +" Time: " + peerids[i]
[1] + " (" + peerids[i][2] +")");
                        }
                        System.out.println("\nYou can choose one to download the file: (Or you
can enter 0 to cancel)");
                        int selection = getSelection(in, 0, peerids.length);

                        //0 means exit. Otherwise start download.
                        if(selection != 0){
                                //Call retrieve and calculate the average download time.
                                startTime = System.nanoTime();
                                byte[] filedata = null;
                                try{
                                        PeerI peerserver = (PeerI)
Naming.lookup(peerids[selection-1][0]);
                                        for(int i=1; i<=looptime; i++){
                                                //Change: now we get FilePackage object instead
of only file data.
                                                FilePackage filepack = peerserver.obtain(filename);
                                                filedata = null;
                                                filedata = filepack.contents;
                                                if(filedata == null){
```

```
                                        System.out.println("ExecuteDownload Error:
This peer keep doesn't this file.");
                                        return;
                                }

                                //Save the file which has just been download.
                                //Change: there is a specific file path for cached
files.
                                File path = new File(cachedfilepath);
                                if(!path.exists()){
                                        try {
                                                path.mkdir();
                                        } catch (Exception e) {

System.out.println("ExecuteDownload Error: Can not create folder " + cachedfilepath +". Please
check.");
                                                return;
                                        }
                                }

                                File downloadFile = new File(cachedfilepath + "/"  +
filename);
                                if(!downloadFile.exists())
                                        try {
                                                downloadFile.createNewFile();
                                        } catch (Exception e) {

System.out.println("ExecuteDownload Error: Can not create file " + filename +". Please check.");
                                                return;
                                        }

                                try {
                                        BufferedOutputStream writer = new
BufferedOutputStream(
                                                        new
FileOutputStream(downloadFile.getAbsolutePath()));
                                        writer.write(filedata, 0, filedata.length);
                                        //Change: Put the file info into cacheFile
map.
                                        CachedInfo info = new
CachedInfo(filepack.info.versionNo, filepack.info.time, filepack.cachedTime, filepack.TTR,
filepack.originIP, filepack.state);
                                        cachedFile.put(filename, info);

/*--------------------end change———————————*/
```

## PeerI.java

```
/*------------------start change---------------------*/
	public enum FileState {
		VALID, INVALID, TTRexpired
	}

	//A class used for record the state of master copied file.
	public class FileInfo implements Serializable{
		private static final long serialVersionUID = 2563450484336495059L;
		int versionNo;
		public Calendar time;

		public FileInfo(int versionNo, Calendar time){
			this.versionNo = versionNo;
			this.time = time; // The last-modify-time for file.
		}

	}

	//A class used for record the state of cached copied file. Need record more info.
	public class CachedInfo implements Serializable{
		private static final long serialVersionUID = 6769040514341619913L;
		public FileInfo info;
		public int TTR;
		public String originIP;
		public FileState state;
		public Calendar cachedTime; //Record this time when it is cached or ensuered by
pulling.

		public CachedInfo(int versionNo, Calendar time, Calendar cachedTime,int TTR,
String originIP, FileState state){
			info = new FileInfo(versionNo, time);
			this.TTR = TTR;
			this.originIP = originIP;
			this.state = state;
			this.cachedTime = cachedTime;
		}
	}

	//A class used for transmitting file with its state situation.
	public class FilePackage extends CachedInfo implements Serializable{
		private static final long serialVersionUID = 9121438525181647004L;
		public byte[] contents; //Record the file contents.

		public FilePackage(int versionNo, Calendar time, Calendar cachedTime, int TTR,
String originIP, FileState state, byte[] contents){
			super(versionNo, time, cachedTime, TTR, originIP, state);
			this.contents = contents;
```

```
                }
        }
/*--------------------end change— — — — — — — — — —*/
```