



UDPeer

Architectuur

v0.1

Robin Wittevrongel & Bruno Windels



Overzicht

UDPeer is een programma dat toelaat bestanden direct van één computer naar een andere te verzenden zonder tussenkomst van een aparte server. Hiervoor wordt UDP multicast gebruikt om andere computers (peers, engels voor “gelijke”, genaamd) te ontdekken. UDPeer ondersteunt het delen van volledige mappen en het bijhouden van een wachtrij.

Protocol

Het protocol bestaat uit twee delen. Het bevraging gedeelte van het protocol laat toe op te vragen welke peers er zijn met welke bestanden en gebruikt multicast en unicast UDP. Het transmissie gedeelte van het protocol wordt gebruikt om de bestanden te verzenden en gebruikt TCP. Beide gedeeltes werken met het concept van commando's met de volgende syntax:

```
COMMANDO <trackid> <argument 1> ... <argument n>
```

Argumenten worden gescheiden door een spatie. Spaties in argumenten worden omgezet naar %20. Procenttekens (%) in een argument worden omgezet naar %25. De trackid is een oplopend nummer dat per peer uniek is en dient als referentienummer naar een bepaald commando. Zo geeft een foutcommando naast een foutbeschrijving en foutnummer ook het trackid mee van het commando waarover de fout gaat:

```
CRAP <trackid> <foutnummer> <foutboodschap> <trackid van foutief commando>
```

Bevragingsgedeelte

Voor het ontdekken van nieuwe peers wordt multicast UDP gebruikt. Op het multicast IP-adres 228.5.6.7 op poort 31415 worden volgende commando's verstuurd:

```
PEER <trackid> <peernaam>  
WELCOME <trackid> <peernaam>  
GONE <trackid>
```

Als een peer online komt verstuurt hij het PEER-commando met zijn naam. Alle andere peers die dit commando ontvangen antwoorden met een WELCOME-commando, eveneens met hun naam als argument. Bij het offline gaan verstuurt een peer het GONE-commando.

Informatie over bestanden, tickets en dergelijke worden verstuurd via unicast UDP op poort 31416 omdat dit enkel twee peers aangaat. Het opvragen van bestandslijsten gebruikt de volgende commando's:

```
LIST <trackid> </pad/>  
FILE <trackid> <trackid lijst commando> <type = E, F of D> <file-i> <file-size>
```

Een peer (A) verstuurt het LIST-commando naar een andere peer (B). Als argument wordt het pad opgegeven waarvan de bestanden moeten teruggegeven worden. UDPeer gebruikt een virtuele bestandsstructuur waar de gebruiken zelf gedeelde mappen kan aan toevoegen. Het eerste niveau van de bestandsstructuur is een door de gebruiker opgegeven naam voor de gedeelde map. A reageert dan ofwel met een foutcommando als de map niet bestaat ofwel met voor ieder bestand in de map met een FILE-commando. Het type voor gewone bestanden is F en voor mappen D. Mappen krijgen geen bestandsgrootte mee. Om de lijst af te sluiten wordt er ook nog een FILE-commando met type E verzonden. Deze krijgt een bestandsnaam noch bestandsgrootte mee.

Als een peer een bestand van een andere peer wil downloaden moet hij eerst een ticket aanvragen voor die download. Zo kan de client veel bestanden op één moment aanvragen en kan de server ze (gedeeltelijk) serieel doorsturen. Iedere peer heeft een instelling hoeveel gelijktijdige uploads hij toestaat. Door het gebruik van tickets kunnen de server en de client synchroon hun wachtrij beheren.

```
REQUEST <trackid> <path>  
TICKET <trackid> <ticketid>  
READY <trackid> <ticketid>  
NEVERMIND <trackid> <ticketid>
```

Als een peer een bestand wil downloaden, verstuurt hij, via unicast UDP, een REQUEST-commando met het pad van het te downloaden bestand. De andere peer antwoordt hierop met een foutcommando of met een TICKET-commando. Dit laatste commando geeft het ticket-id terug waarmee deze download vanaf nu zal geïdentificeerd worden. Als de verzendende peer klaar is om het bestand te verzenden laat hij dit weten aan de andere peer door middel van het READY-commando. Als een van de peers de download wil annuleren alvorens deze begonnen is kan het ticket verwijderd worden door het NEVERMIND-commando. Verder is er nog het ROGER-commando om te bevestigen dat een ander commando goed ontvangen is:

```
ROGER <trackid> <trackid van bevestigd commando>
```

Transmissiegedeelte

Bij het ontvangen van het READY-commando zal de client een TCP socket openen op poort 31415 naar de andere peer. Alle verzonden commando's via TCP worden gesplits door het newline karakter. Na het openen van de socket verstuurt de client het volgende commando:

```
GIME <trackid> <ticketid>
```

De server antwoordt hierop met een foutcommando als het ticket niet bestaat of nog niet gestart is. Als dit niet het geval is stuurt de server het volgende commando:

```
OKAY <trackid> <content length>
```

Het OKAY-commando laat de client weten dat na dit commando de data van het aangevraagde bestand volgt. Het geeft ook aan hoeveel bytes er zullen verzonden worden. Om de overdracht te annuleren wordt de socket gewoon gesloten.

Ontwerp

Controller

De controller is een singleton en biedt alle functionaliteit van het domein aan die de GUI nodig heeft. De methoden `startServer` en `stopServer` starten alle threads die op netwerkverkeer moeten luisteren:

- `FileServer` voor het luisteren op TCP
- `PeerServer` voor het luisteren op multicast UDP
- `ShareServer` voor het luisteren op unicast UDP

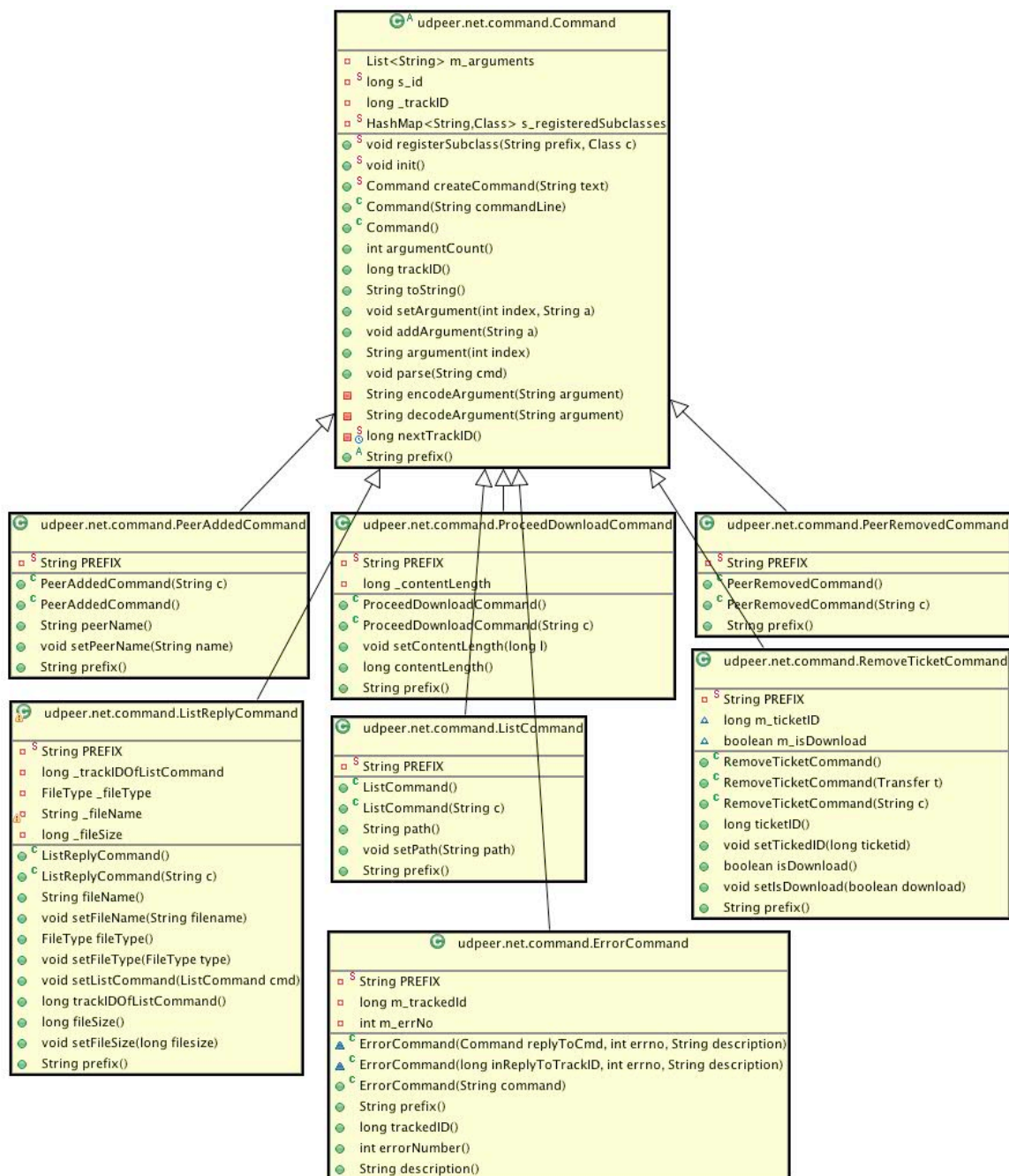
De controller delegeert het opvragen van bestandslijst aan `ShareManager`. Het aan- en af melden wordt doorgegeven aan `PeerManager` en het downloaden van een bestand geeft de controller door aan `ShareManager` (voor het ticket gedeelte) en `FileClient` (voor het ontvangen).

```
udpeer.domain.UDPeerController

boolean isConnected()
UDPeerController instance()
void init()
UDPeerController()
ShareManager shareManager()
PeerManager peerManager()
Set<Peer> peers()
void download(Transfer t)
void listFiles(Peer p, String filepath, FileListListener l)
void startServer()
void stopServer()
void addUploadListener(UploadListener l)
void removeUploadListener(UploadListener l)
void addPeerListener(PeerListener l)
void removePeerListener(PeerListener l)
void addShareFolder(String name, String path)
void shutdown()
int getMaxUploads()
void setMaxUploads(int max)
void cancelDownload(Transfer t)
void cancelUpload(Transfer t)
void setPreferenceForKey(String key, String preference)
String preferenceForKey(String key)
void storePrefs()
void peerAdded(Peer p)
void peerRemoved(Peer p)
void startDownload(Transfer t)
void notifyNewUpload(Transfer t)
UploadScheduler uploadScheduler()
DownloadManager downloadManager()
void requestTicketID(Transfer t)
void transferFinished(Transfer t)
void cancelAllTransfers()
String settingsFilePath()
void ticketReady(Transfer t)
void sendCancelTicket(Transfer t)
void receiveCancelTicket(Peer p, long ticketid, boolean isDownload)
void cancelUploadsForPeer(Peer p)
```

Commando

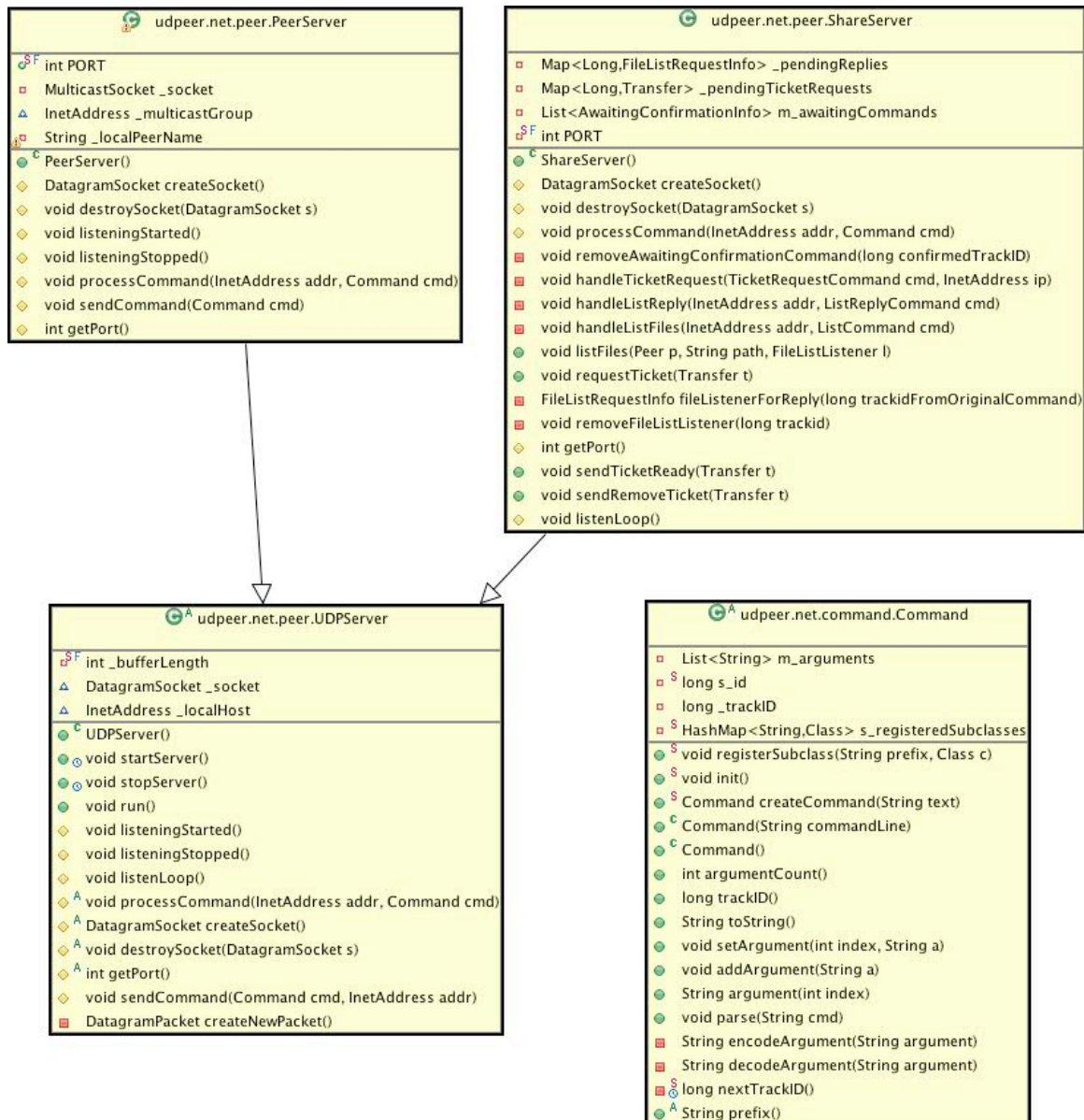
Voor het eenvoudig versturen en ontvangen van commando's is er de Command klasse met al zijn subklassen. Commando zorgt ervoor dat alle argumenten kunnen ingesteld en opgevraagd worden en kan commandotekst parsen. Subklassen zullen een specifiek commando voorstellen en de argumenten omzetten naar de juiste datatypes en hiervoor ook getters en setters voorzien. Zowel UDPServer (de superklasse van PeerServer en ShareServer) als TCPFileTransfer (de superklasse voor uploads en downloads) bieden een methode processCommand(Command c) en sendCommand(Command c) voor het verwerken van ontvangen commando's en het versturen van nieuwe commando's.



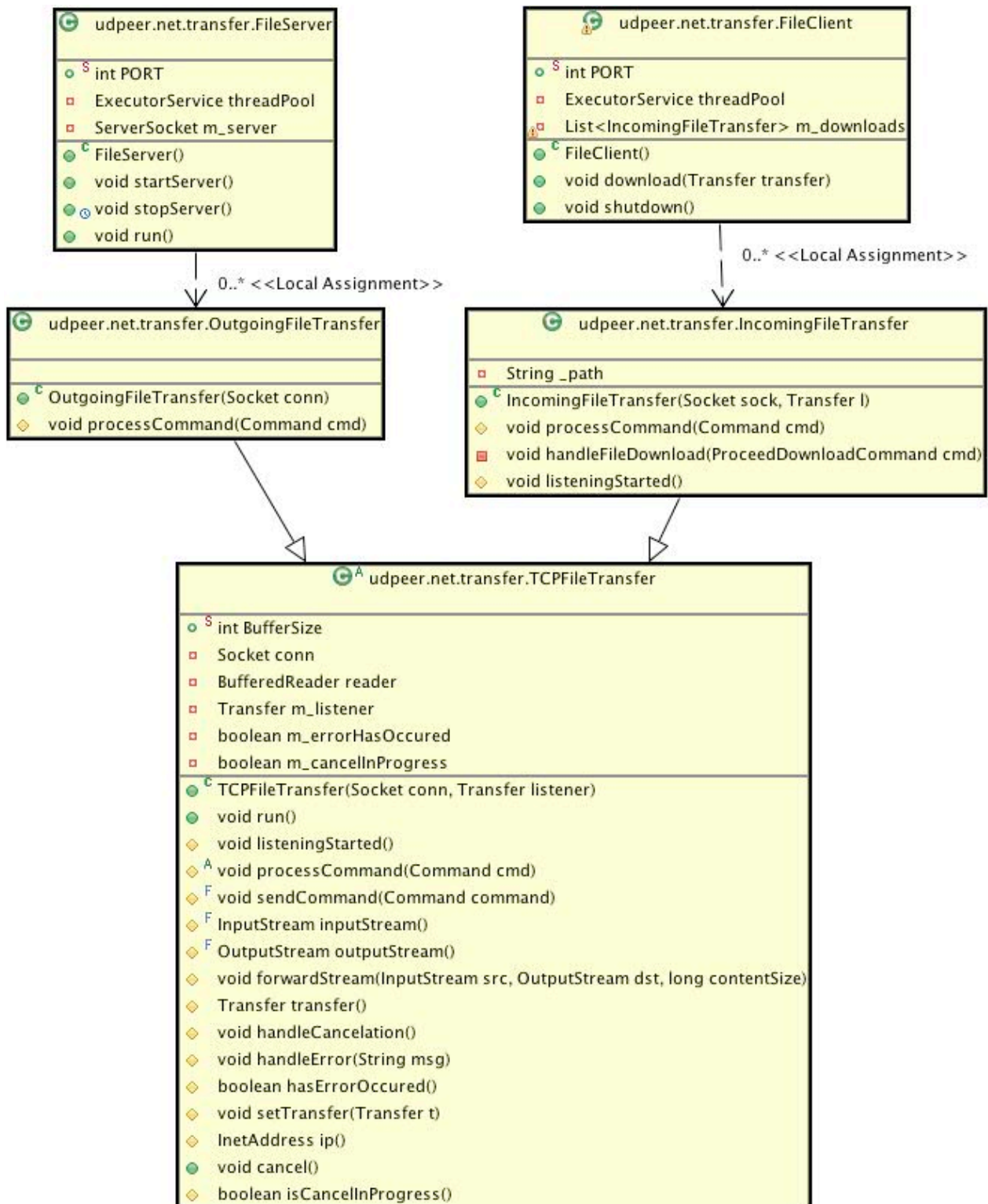
UDPServer

Omdat het afhandelen van het multicast en unicast UDP verkeer zoveel gemene code heeft erven PeerServer en ShareServer van UDPServer. Voor de zaken die niet gemeenschappelijk zijn zijn er polymorfisch methodes:

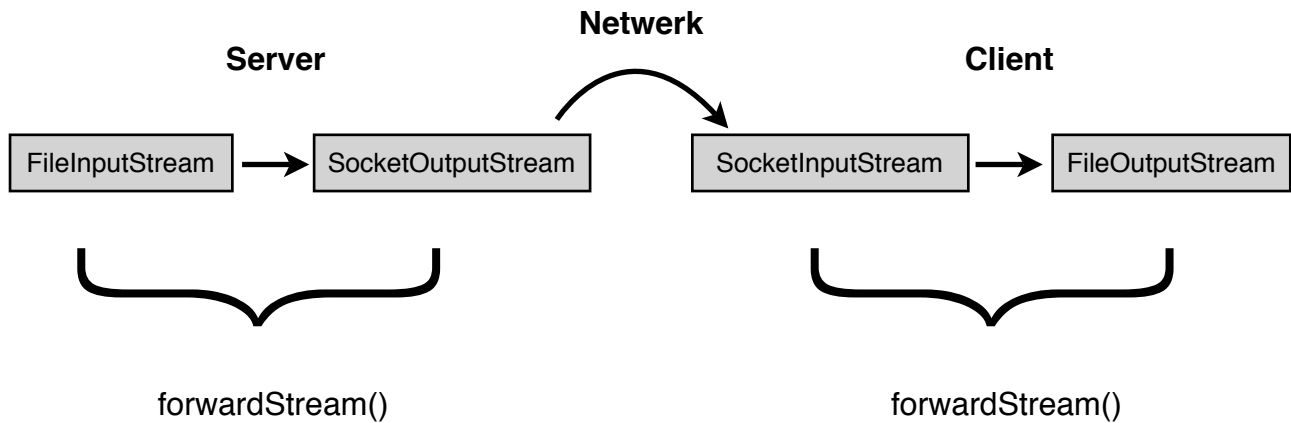
- het aanmaken en sluiten van de DatagramSocket gebeurt in de subklassen via createSocket() en destroySocket(DatagramSocket s).
- Het verwerken van ontvangen commando's wordt gedelegeerd aan de subklassen in de methode processCommand(Command c).
- Omdat ShareServer niet bevestigde Commando's opnieuw moet versturen moet hij toegang hebben tot de luisterlus op de socket. Daarom wordt in iedere iteratie listenLoop() aangeroepen.



TCPFileTransfer, FileServer en FileClient



Het versturen en het ontvangen van bestanden heeft ook veel gemene code. Daarom gebeurt het grootste deel van het versturen en ontvangen in de superklasse TCPFileTransfer. In de subklassen OutgoingFileTransfer (voor uploads) en IncomingFileTransfer worden de bestanden om te lezen en schrijven geopend en de commando's afgehandeld. Het versturen en ontvangen van bestanden gebruikt dezelfde code:



Iedere TCPFileTransfer draait in zijn eigen thread. OutgoingFileTransfer's worden aangemaakt en gestart door FileServer die luistert op de ServerSocket.

IncomingFileTransfer's worden aangemaakt en gestart door FileClient als de methode download() aangeroepen wordt.