Programming Assignment #1

Development Environment

- **macOS Big Sur** 11.0.1 20B29 x86_64
- **java 12.0.2** 2019-07-16

Build and Run

```
$ cd /path/to/code/ene4019
$ javac FTPServer.java
$ javac FTPClient.java
$ java FTPServer &
$ java FTPClient
```

Implementation

JAVA를 이용해, 간단한 FTP server와 client를 구현한다.

핵심적인 부분들은 모두 `ftp` 폴더 내에 구현되었으며, `FTPServer.java` 와 `FTPClient.java` 는 이를 실행시키는 역할만 한다.

지금부터 각 패키지의 중요한 부분만 간단히 설명한다.

`ftp` package

class DataChunkC2S

Client-to-Server data chunk를 나타낸다.

```
public DataChunkC2S(byte[] bytes)
```

다른 machine과의 통신을 통해 수신한 byte array를 다시 해석가능한 형태로 deserialize한다.

```
public void writeBytes(DataOutputStream dataOutputStream) throws IOException
```

Data chunk를 serialize하여, 다른 machine에 연결된 output stream으로 전달한다.

class DataChunkS2C

Server-to-Client data chunk를 나타낸다. `DataChunkS2C`와 큰 차이가 없으므로 설명을 생략한다.

enum ReturnCode

서버의 응답 코드를 나타낸다. 각 코드마다 `int`형 고유번호를 가지고 있으며, 숫자는 대부분 위키피디아에서 가져와 사용했다.

class Response

서버의 응답 메시지를 나타낸다. 위에서 설명한 `enum ReturnCode`를 포함하고 있다. 모든 Response는 리턴 코드 다음에 띄어쓰기 한 칸이 있으며, 그 다음에 메시지가 들어간다. 메시지의 끝에는 최소 두 개의 newline이 들어간다. 이 클래스는 이러한 구조를 정확히 유지하는 것을 돕는다.

```
public Response(String responseStr)
```

`String`으로 된 응답을 client가 해석 가능한 형태로 deserialize한다. `class Response`는 내부적으로 hash map을 가지고 있는데, 이를 통해 int형으로 된 return code를 `enum ReturnCode`로 빠르게 변환할 수 있다.

```
public String toString()
```

response의 전송이 가능하도록 serialize한다. Response format을 잘 맞출 수 있도록 newline이 두 개 이상 존재하는지를 확인한다.

`ftp.client` package

class Client

클라이언트 프로그램.

```
protected class ConnectionTerminatedException extends Exception
```

Server와의 연결이 끊기는 경우에 발생시키는 exception이다. 아직은 별도의 대응을 하지 않았지만, 이후 재접속을 시도하는 등 다양한 처리가 가능할 것이다.

```
public void start(String host, int cmdPort, int dataPort) throws IOException
```

클라이언트 프로그램을 실행시킨다.

socket과 reader를 만드는 등 서버에 연결하는 작업과 함께, 기본적인 세팅들을 한다. 서버에 초기부터 접속하지 못하는 경우 `IOException` 을 발생시키며, `FTPClient` 에서는 이를 catch하여 재접속을 시도하는 데 사용한다. 정상적으로 서버에 접속된 경우, 사용자로부터 request를 받고, 이를 server에 보낸 후, response를 받는 것을 반복한다. `GET`과 `PUT` 명령어의 경우 data channel을 여는 등 좀 더 복잡한 처리가 필요하므로 별도 함수를 통해 서버와 통신한다.

```
protected Response getResponse() throws IOException, ConnectionTerminatedException
```

server에서 response를 받아 온다. Response의 format에 맞도록, newline이 두 개 읽힐 때까지 command channel에서 문자열을 읽어 온다. 읽기를 마지고 나면 이를 해석 가능한 형태로 deserialize해 return한다.

```
protected int _get(String request) throws IOException, ConnectionTerminatedException
```

Server로부터 파일을 받아 오는 명령어인 `GET`을 처리하기 위한 별도 method이다. 이 함수는 다음과 같은 순서로 작동한다.

1. 먼저, 서버에서 가져오고자 하는 파일명을 서버에게 알려 준다.
2. 서버의 응답을 받은 다음, 실패 응답이 온 경우 종료한다.
3. 다음으로는 target의 정보(파일 길이)를 얻어온다.
4. Data channel을 열고 이를 통해 통신을 시작하며, 동시에 파일을 저장한다.

```
protected int _put(String request) throws IOException, ConnectionTerminatedException
```

Server로 파일을 업로드하는 명령어인 `PUT`을 처리하기 위한 별도 method이다. 다음과 같은 순서로 작동한다.

1. 서버에 요청하기 전에 자신에게 그 파일이 존재하는 것인지 확인한 다음, 요청을 보낸다.
2. 응답을 확인해, 실패 응답이 온 경우 종료한다.
3. target의 정보(파일 길이)를 보낸다.
4. Data channel을 열고 파일을 보낸다.

`ftp.server` package

class Server

서버 프로그램.

```
public void start(int cmdPort, int dataPort) throws IOException
```

서버 프로그램을 시작시킨다. command channel과 data channel을 위한 server socket을 초기화시키며, 새로운 client가 도착할 때마다 `Server::ClientManager` 를 만들고 실행시킨다. 한 client의 접속이 끝나면 다른 client가 올 때까지 계속 기다린다.

class Server::ClientManager

한 명의 client를 담당하는 클래스로, Server class의 inner class이다. 각 client에 대한 정보들도 담고 있다.

```
public void start(Socket cmdSocket, ServerSocket serverDataSocket)
```

client 요청 처리를 시작한다. IO stream을 세팅하는 등 기본적인 작업들을 하고 난 다음, request를 읽고 처리하는 것을 계속해서 반복한다.

```
protected void processRequest(String[] request) throws ConnectionDownException, IOException
```

request를 처리한다. request를 실제로 처리하는 것은 명령어마다 따로 있는 method들이지만, 이들을 적당하게 호출하는 역할을 하는 것이 이 method다. `Server` 클래스 내에 명령어 문자열을 함수 호출로 mapping해주는 자료구조가 있는데, 이를 활용하게 된다.