1. Server design.
   - The server is based on EchoServer and EchoHandler multithreading. In EchoHandler the run() method is override and the server is implemented within it:
   - The server first reads the input line and processes the lines using requestProcessing() function. Inside the function it stores the file path, and checks the “Range:” header and stores the range values in long[] fileRange field accordingly.
   - Then the server gets the current date. We create a ZoneId object using the string "GMT" to represent the GMT time zone, then we applied the pattern "EEE, dd MMM yyyy HH:mm:ss z" and the English locale.
   - Then the server checks the file’s existence using the filePath field. If it doesn’t exist, the server responds with “404 Not found”, if it exists, it determines to send “200 OK” or “206 Partial Content” response by checking the fileRange field’s value to be instantiated or not.
   - The serveFile() method is for “206 partial content” response. It skips the 0~start part, and sends through the end.
   - Finally, close all streams and the client.
2. Multithreading

   Multithreading is accomplished using Java thread, start() and run(). In our test, we can handle 500 threads in about 2 minutes.
3. Libraries

```
import java.io.*;
import java.net.Socket;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.HashMap;
import java.util.Locale;
import java.util.Map;
```

4. Extra capacities

   We tried to cache the files in the server memories so that the threads don’t need to retrieve the fileInputStream so many times. However, we abandoned this idea for fear of too much memory usage, and the fileInputStream is pretty effected in that it reads a buffer of custom size and write it out directly, thus caching the file is not really necessary.
5. Instruction of Our Code:

   We use the java language to implement our project1. EchoServer.java contains the main function and EchoHandler.java contains all the functions we created. They are all included in the folder “src”. The folder "content" contains all the files we use to test our server. The “build.xml” file is the ANT script we use to compile the java code. When using our code, you need to go to our folder and open terminal. Type in “ant” command to compile. Then you could see two files, EchoServer.class and EchoClient.class. After that, just type the command “java EchoServer”, then we could see the server is now running and waiting for the clients. If you are testing the server, just go to the chrome browser and type in the command http://localhost:10007/image.jpg , you can also change the port when running the code, for

example, run “java EchoServer 8887” command, then you could use the port 8887.