XML Error

When I run the code, I don't seem to be getting an XML response in the GETClient. So the first thing I did was to try and debug the error in the GETClient to try and see where the error was coming from. The method I used to achieve this was to use print statements to the console to check where the code stops.

I noticed that I seemed to successfully create a socket, and send a GET request to the server. However, it appears that I was not receiving an XML response from the server as the print statement was not printing after these first two lines.

```
ois = new ObjectInputStream(socket.getInputStream());
Packet responsePacket = (Packet) ois.readObject();
System.out.println("HERE 3 - Successfully recieved XML response from server");
```

It appears that the code seemed to be hanging which indicated that the error was coming from the AggregationSever. So I went over to the AggregationServer and began adding in the debugging comments to find more information regarding the point of error.

The steps that need to be taken by the Aggregation Server are as followed :

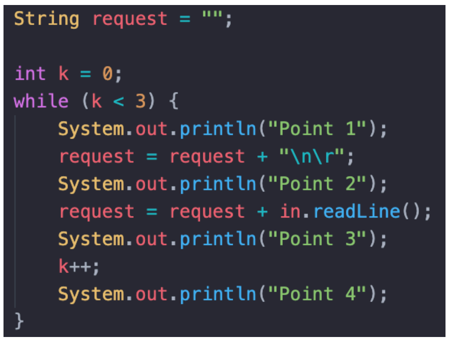
1. Accept the incoming request.
2. Read the client's request.
3. Process the request.
4. Send the request back to the client.

It appears that the Aggregation Server seemed to be accepting an incoming connection correctly after running :

```
Socket socket = server.accept();
```

However, the issue seemed to be related to the BufferedReader where it reads the client's request, and more specifically, in the loop where it constructs the request. During our consultation sessions, Roshan had a look at the README file and the author of the code mentioned that he encountered an error with the BufferedReader that prevented his code from working and he was unable to resolve it.

I added debugging print statements between each line to see where the error was coming from (Point 1 to 4).



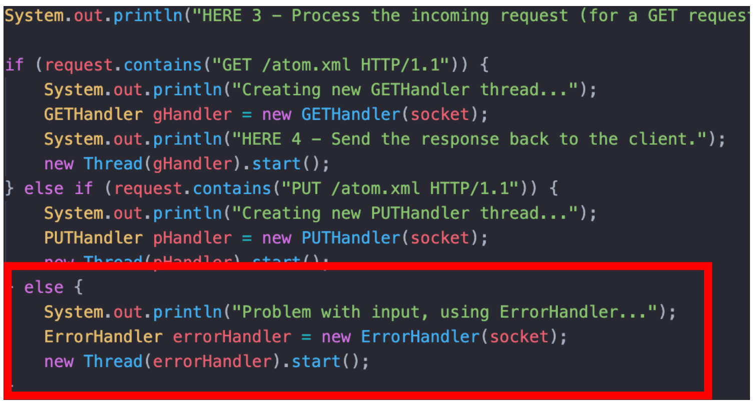
```
String request = "";

int k = 0;
while (k < 3) {
    System.out.println("Point 1");
    request = request + "\n\r";
    System.out.println("Point 2");
    request = request + in.readLine();
    System.out.println("Point 3");
    k++;
    System.out.println("Point 4");
}
```

It seemed that the code would hang after Point 2 which meant that the error was coming from the line :

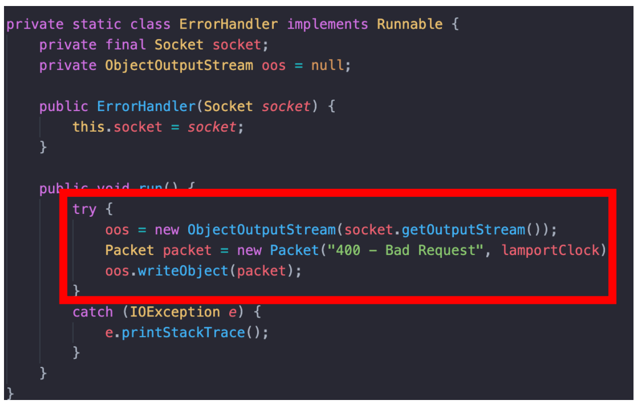
```
request = request + in.readLine();
```

Which made sense as this line uses the BufferedReader.

As an experiment I wanted to try and test the program to see what would happen if I removed this line and essentially create a request that contained no useful information. Funny enough, the program actually seemed to work. Obviously, it did not do anything useful with the request and did not provide an XML output. However, it seemed to attempt to process the request (which resulted in an ErrorHandler as expected) and send a response back to the client (which resulted in a 400-Bad Request in the GETClient).



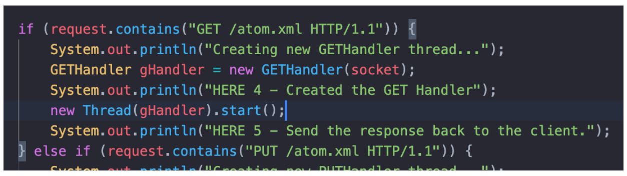
```
System.out.println("HERE 3 - Process the incoming request (for a GET reques

if (request.contains("GET /atom.xml HTTP/1.1")) {
    System.out.println("Creating new GETHandler thread...");
    GETHandler gHandler = new GETHandler(socket);
    System.out.println("HERE 4 - Send the response back to the client.");
    new Thread(gHandler).start();
} else if (request.contains("PUT /atom.xml HTTP/1.1")) {
    System.out.println("Creating new PUTHandler thread...");
    PUTHandler pHandler = new PUTHandler(socket);
    new Thread(pHandler).start();
  else {
    System.out.println("Problem with input, using ErrorHandler...");
    ErrorHandler errorHandler = new ErrorHandler(socket);
    new Thread(errorHandler).start();
```



```
private static class ErrorHandler implements Runnable {
    private final Socket socket;
    private ObjectOutputStream oos = null;

    public ErrorHandler(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try {
            oos = new ObjectOutputStream(socket.getOutputStream());
            Packet packet = new Packet("400 - Bad Request", lamportClock)
            oos.writeObject(packet);
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Since there was no valid information, it resulted in an error. However, it seemed to process this response correctly and send the response back to the client. So, it appears that the server is communicating with the client. However, it is not constructing a proper request. Therefore, I then attempted to construct a dummy request to see if I could trick the Aggregation Server into sending a GET response back to the client. I noticed that the code only checks to see if the request contains **"`GET /atom.xml HTTP/1.1`"** within it's string.

Therefore, I commented out the loop which constructs the request in entirety and made it so that the request variable only contained **"`GET /atom.xml HTTP/1.1`"** and ran the code to see what would happen. For the sake of debugging the code, I added in a few extra print statements to see how the code executes.



```
if (request.contains("GET /atom.xml HTTP/1.1")) {
    System.out.println("Creating new GETHandler thread...");
    GETHandler gHandler = new GETHandler(socket);
    System.out.println("HERE 4 - Created the GET Handler");
    new Thread(gHandler).start();
    System.out.println("HERE 5 - Send the response back to the client.");
} else if (request.contains("PUT /atom.xml HTTP/1.1")) {
    System.out.println("Creating new PUTHandler thread...");
```

Funny enough, it seemed to run the code, create a new GET Handler for the socket, and send the response back to the client. However, this obviously did not solve my error because the information within the request was essentially pointless and did not contain the information required to actually process it.

This is where I have been stumped and have been unsure how to solve the problem. It feels close but I'm not sure what to do about the BufferedReader. I also understand that there is an error in the way the program just runs in general.

Originally the client expects a response before if has even sent a request. It should instead send a GET request when the GETClient connects to the server. Once the request is sent, the client should then wait for the server to send an XML response back to the client. I believe this is the foundational error that Roshan mentioned to me in our consulting session.

However, upon inspection, it also appears that the Content Server also is not able to read and return the file data meaning that the PUT request is not being sent to the Aggregation Server after the following line :

```
ois = new ObjectInputStream(socket.getInputStream());
```

So I moved this line to be under the first lamportClock increment after the outputWriter executes and I noticed that the in.readLine() started to work to some extent.

I then noticed that in the contentServer, the line :

```
String xmlOut = xmlFactory.buildXML(inputFile, ID);
```

Had the parameters flipped the wrong way to what the buildXML() function in the XMLFactoryImplementation expects. This also helped solve some issues relating to compiling the Content Server as now the inputs work correctly. However, I became suspicious that this reflected the idea that the author did not get the code to work as I mentioned before.