Building a Multi-Threaded Web Server

Part 1. Understand the given Multi-threaded Echo Server Code in Java

Below, we will go through the code for the implementation of a multi-threaded Echo Server (capable of establishing TCP connections with multiple clients simultaneously and echoing text in upper case).

In the multi-threaded server below, the processing of each incoming request will take place inside a separate thread of execution. This allows the server to service multiple clients in parallel, When we create a new thread of execution, we need to pass to the Thread's constructor an instance of some class that implements the `Runnable` interface. This is the reason that we define a separate class called `serviceRequest`. The structure of the multi-threaded echo server is shown below:

Note: More information about creating threads in Java can be found here:
https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html

```
import java.io.* ;
import java.net.* ;
import java.util.* ;

public class incomingRequest {
    public static void main(String argv[]) throws Exception {

    // Get the port number from the command line.
      int port = (new Integer(argv[0])).intValue();

      // Establish the listen socket.
      ServerSocket socket = new ServerSocket(port);

      // Process incoming requests in an infinite loop.
      while (true) {
          // Listen for a TCP connection request.
          Socket connection = socket.accept();

          // Construct an object to process the incoming request
          serveRequest request = new serveRequest(connection);

          // Create a new thread to process the request.
          Thread thread = new Thread(request);

          // Start the thread.
          thread.start();
      }
    }
}
```

```
import java.io.* ;
import java.net.* ;
import java.util.* ;

public class serveRequest implements Runnable {

    Socket socket;

    // Constructor
    public serveRequest(Socket socket) throws Exception {
        this.socket = socket;
    }

    // Implement the run() method of the Runnable interface.
    public void run() {
    try {
        processRequest();
    }   catch (Exception e) {
        System.out.println(e);
    }
    }

    private void processRequest() throws Exception {

    // Get a reference to the socket's input and output streams.
    InputStream is = socket.getInputStream();
    DataOutputStream os = new
      DataOutputStream(socket.getOutputStream());

    // Set up input streams
    BufferedReader br = new BufferedReader(new InputStreamReader(is));
    while (true) {

      // Get the incoming message from the client (read from socket)
      String msg = br.readLine();

      //Print message received from client
      System.out.println("Received from client: ");
      System.out.println(msg);

      //convert message to upper case
      String outputMsg = msg.toUpperCase();

      //Send modified msg back to client (write to socket)
      os.writeBytes(outputMsg);
      os.writeBytes("\r\n");
      System.out.println("Sent to client: ");

      }

   }

}
```

Test the above Multi-threaded echo server by running multiple TCP Clients (use code posted earlier on Moodle).

Once you are comfortable with how the above server operates, you can proceed with the Multi-threaded Web Server Assignment.

Specifically, your Web server will (i) create a connection socket when contacted by a client (browser); (ii) receive the HTTP request from this connection; (iii) parse the request to determine the specific file being requested; (iv) get the requested file from the server's file system; (v) create an HTTP response message consisting of the requested file preceded by header lines; and (vi) send the response over the TCP connection to the requesting browser. If a browser requests a file that is not present in your server, your server should return a " 404 Not Found" error message.

Note: The sample web page called samplePage.html which has 4 embedded objects (3 jpeg files and one mp3 file) are all supplied to you and uploaded on Moodle.