```
/*client_socket - HTTP client implementation to connect to a specified server on predefined
port.
 * This can be implemented for standard HTTP GET and PUT requests.
 * Author   - Kiran C Shettar
 * Email_ID - KiranChandrashekhar_Shettar@student.uml.edu
 */

import java.io.*;
import java.net.*;

public class client_socket
{
    //declaration of global variables
    private Socket clientSocket = null;
    private BufferedInputStream client_input;
    private BufferedOutputStream client_output;
    //GET and PUT are the method names
    //web site name will be host_name
    private String host_name,method_name;
    private String object_path;
    private int port;

    //initializing global variables (constructor)
    client_socket(String host1, int port1, String method, String path)
    {
        this.host_name = host1.trim();
        this.port = port1;
        this.method_name = method.trim();
        this.object_path = path;
        init_socket();
    }

    //verifying the arguments
    //creating an instance of client_socket class
    public static void main(String[] args)
    {
        String host = args[0];
        int port1 = Integer.parseInt(args[1]);
        String method = args[2];
        String path = (args[3]);

        //checking the conditions here
        if (host == null)
        {
            System.out.println("Error: Declare the Host Name/ Host IP in Arguments");
        }
        else if (port1 <= 0)
        {
            System.out.println("Error: Declare Port number in Arguments");
        }
        else if (method == null)
        {
            System.out.println("Error: Declare a Method_name in Arguments (GET/PUT)");
        }
        else if (path.toString() == null)
        {
            System.out.println("Error: Declare the Path/File name in Arguments (XYZ.html)");
        }

        client_socket client = new client_socket(host, port1, method, path);
    }

    //establishing socket connection between client & server
    //initializing the input and output streams
    private void init_socket()
    {
```

```
        try
        {
            String request = null;
            clientSocket = new Socket(host_name, port);
            System.out.println("CONNECTION ESTABLISHED: client & server");
            System.out.println("Client IP : "+clientSocket.getLocalAddress());
            System.out.println("Server IP : "+clientSocket.getInetAddress()+" :
            "+clientSocket.getPort());

            client_input = new BufferedInputStream(clientSocket.getInputStream());
            client_output = new BufferedOutputStream(clientSocket.getOutputStream());
            System.out.println("\nCONNECTION INITIALIZED: client Input/Output streams");

            //function call depending on the HTTP method passed as argument
            if(method_name.equalsIgnoreCase("get"))
            {
                get_method();
            }
            else if(method_name.equalsIgnoreCase("put"))
            {
                put_method();
            }
            else
            {
                request = method_name+ " " + object_path + " " + "HTTP/1.0";
                sendRequest(client_output,request);
            }
            clientSocket.close();
        }

        catch (UnknownHostException e)
        {
            e.printStackTrace();
        }

        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    //function handling for sending the client request
    private static void sendRequest(BufferedOutputStream output, String request) throws
    Exception
    {
        System.out.println();
        System.out.println("CLIENT REQUEST: ");
        System.out.println(request);
        byte [] buff = request.getBytes();

        output.write(buff);
        output.flush();
    }

    //function handling for reading the server's response
    private static void readResponse(BufferedInputStream in) throws Exception
    {
        byte[] contents = new byte[2048];
        int bytesRead = 0;
        System.out.println("SERVER'S RESPORNSE: ");

        while ((bytesRead = in.read(contents)) != -1)
        {
            System.out.println(new String(contents, 0, bytesRead));
        }
    }
```

```
    //function handling for message exchange between client and server
    //client GET request
    private void get_method () throws Exception
    {
        String URL = method_name + " /" + object_path + " " + "HTTP/1.0" + "\r\n\r\n";

        try
        {
            sendRequest(client_output, URL);
            readResponse(client_input);
        }

        catch(Exception e)
        {
            e.printStackTrace();
        }

        finally
        {
            //CLOSING socket,input stream, output stream
            client_output.close();
            client_input.close();
            if(clientSocket.isClosed() != true)
            clientSocket.close();
        }
    }

    //function handling used for constructing and sending
    //client PUT request
    private void put_method() throws Exception
    {
        int count = 0;
        FileInputStream output_file = null;
        BufferedInputStream buffer_input = null;
        String URL = method_name + " " + object_path;

        try
        {
            sendRequest(client_output, URL);
            //reading the data in file and sending it over socket as byte
            File fp = new File(object_path);
            long length = fp.length();
            if(length > Integer.MAX_VALUE)
            System.out.println("LARGE FILE");

            byte[] buffr = new byte[(int)length];
            output_file = new FileInputStream(object_path);
            System.out.println("File size = " + output_file.available());

            buffer_input = new BufferedInputStream(output_file);

            while((count = buffer_input.read(buffr)) > 0)
            {
                client_output.write(buffr, 0, count);

            }
            clientSocket.shutdownOutput();
            readResponse(client_input);
        }

        catch(Exception e)
        {
            e.printStackTrace();
        }
```

```
        finally
        {
            //CLOSING END socket, file stream, input stream, output stream
            buffer_input.close();
            client_output.close();
            client_input.close();
            output_file.close();
            if(clientSocket.isClosed() != true)
            clientSocket.close();
        }
    }
}
```