

Milwaukee School of Engineering CS496 – Networking Protocols

Lab5 – XML-RPC Client/Server

Student: Marek Handl

Date: February 22, 2007

➤ SOURCE CODE

```
// ***** Server.java *****
/*
 * CS496 - Lab5 - XML-RPC Server/Client - Server Module
 * author: Marek Handl
 * date: February 22, 2007
 * *****
 * The project consists of two programs - a Server handling xml-rpc requests and a Client sending
the requests.
 *
 * This is the central server part. It listens on chosen port for incoming connections.
 * For each connection a new thread (ServerThread) is created and executed.
 */

import java.io.IOException;
import java.net.InetAddress;
import java.net.ServerSocket;

/* Class Client is the only class in the base part of the Server program
 * See file documentation for functionality description
 */
public class Server {

    /**
     * CS496 - Lab5 - XML-RPC Server/Client - Server Module
     * @author: Marek Handl
     * @date: February 22, 2007
     * *****
     * main function of the Server (listening part)
     * parses command parameters, listens on chosen port, creates threads for incoming
connections
     *
     * @param args      - input, command line arguments
     */
    public static void main(String[] args) throws IOException {

        // implicit parameters
        int port = 80;
        String path = "/RPC";

        // parsing of command line parameters
        if ( args.length > 0 ) {
            if ( args[0].equals("-h") ) { // display help
                System.out.println( "Usage: Server [ port_number | RPCpath ]" );
                System.exit(1);
            }
            else {
                // parse port number
                try {
                    port = Integer.parseInt( args[0].trim() );
                }
            }
        }
    }
}
```

```

        catch (Exception e) {
            System.out.println( "Wrong parameters!" );
            System.out.println( "Usage: Server [ port_number | RPCpath ]" );
            System.exit(1);
        }
    }
}
boolean alwaysTrue = true;

try {
    // start listening
    ServerSocket srvsoc = new ServerSocket( port );
    System.out.println( "Server running." );
    System.out.println( "Server IP address: " + InetAddress.getLocalHost() );
    System.out.println( "Listening on port: " + port );
    System.out.println( "RPC path: " + path );

    // wait for clients to connect and create a new thread for each one
    while ( alwaysTrue ) {

        new ServerThread( srvsoc.accept(), path ).start();

    }

    // stop listening and exit
    srvsoc.close();
    System.out.println( "Server shut down" );

}
catch (Exception e ) {
    System.out.println( "Something went wrong - server!!! " + e.getMessage() );
}
}
}

```

```

// ***** ServerThread.java *****
/*
 * CS496 - Lab5 - XML-RPC Server/Client - Server Module
 * author: Marek Handl
 * date: February 22, 2007
 * *****
 * The project consists of two programs - a Server handling xml-rpc requests and a Client sending
the requests.
 *
 * This is the thread part of the server, that handles only one client.
 * Parses client's request, calls appropriate computation methods, sends result back to the client
in xml-rpc format.
 * In case of an error, fault message is generated and sent.
 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.*;

/* Class Client is the only class in the user thread part of Server program
 * It is a thread class
 * See file documentation for functionality description
 */
public class ServerThread extends Thread {

    private String rpcPath; // root directory of the web server
    private boolean debug = false; // display detailed information about connection?

```

```

private Socket socket; // socket connection with the client
private String httpType; // HTTP/1.0 or HTTP/1.1
private PrintWriter out; // socket output
private BufferedReader in; // socket input
private String headers = "Connection: Close\r\n" +
                        "Server: RPCServer/0.1\r\n" +
                        "Content-Type: text/xml\r\n"; // common headers

private String xml = ""; // here the response xml is build
/**
 * CS496 - Lab5 - XML-RPC Server/Client - Server Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * simple constructor
 *
 * @param socket - input, used for connection
 * @param path - input, root directory
 */
public ServerThread( Socket socket, String path ) {
    super("ServerThread");
    this.socket = socket;
    this.rpcPath = path;
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Server Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * sends fault response to the Client with description of the error
 * response is in xml-rpc format
 */
private void faultResponse( String message ) throws IOException {
    if ( message.equals("") ) // default error message
        message = "Method call error!";
    System.out.println( "ERROR - " + message );
    xml = "<?xml version=\"1.0\"?>\r\n" +
        "<methodResponse>\r\n" +
        "<fault>\r\n" +
        "<value>\r\n" +
        "<struct>\r\n" +
        "<member>\r\n" +
        "<name>faultCode</name>\r\n" +
        "<value><int>1</int></value>\r\n" +
        "</member>\r\n" +
        "<member>\r\n" +
        "<name>faultString</name>\r\n" +
        "<value><string>" + message +
        "</string></value>\r\n" +
        "</member>\r\n" +
        "</struct>\r\n" +
        "</value>\r\n" +
        "</fault>\r\n" +
        "</methodResponse>\r\n";

    // error in rpc calling is still an http OK message
    String response = httpType + " 200 OK\r\n" +
        "Content-Length: " + xml.length() + "\r\n" + headers +
        "\r\n";
    response += xml;

    if ( debug ) System.out.println( "\n" + response );

    out.println( response ); // send response to the client

    closeConnection();
}
/**

```

```

* CS496 - Lab5 - XML-RPC Server/Client - Server Module
* @author: Marek Handl
* @date: February 22, 2007
* *****
* creates first lines of xml-rpc response
*/
private void createResponse() {
    xml = "<?xml version=\"1.0\"?>\r\n" +
        " <methodResponse>\r\n" +
        " <params>\r\n";
}

/**
* CS496 - Lab5 - XML-RPC Server/Client - Server Module
* @author: Marek Handl
* @date: February 22, 2007
* *****
* encodes parameter into xml-rpc format
*
* @param parameter      - Object, (int, double, boolean or String) that carry the result
value
*/
private void addParameter( Object parameter ) {
    // Integer
    if ( parameter.getClass().getName().equals( Integer.class.getName() ) ) {
        xml += " <param>\r\n" +
            " <value><i4>" + (Integer) parameter + "</i4></value>\r\n" +
            " </param>\r\n";
    }
    // Double
    else if ( parameter.getClass().getName().equals( Double.class.getName() ) ) {
        xml += " <param>\r\n" +
            " <value><double>" + (Double) parameter +
"</double></value>\r\n" +
            " </param>\r\n";
    }
    // String
    else if ( parameter.getClass().getName().equals( String.class.getName() ) ) {
        xml += " <param>\r\n" +
            " <value><string>" + (String) parameter +
"</string></value>\r\n" +
            " </param>\r\n";
    }
    // Boolean
    else if ( parameter.getClass().getName().equals( Boolean.class.getName() ) ) {
        xml += " <param>\r\n" +
            " <value><boolean>" + (Boolean) parameter +
"</boolean></value>\r\n" +
            " </param>\r\n";
    }
    // other types are not supported
    else {
        System.out.println( "ERROR - Unknown parameter type!" );
        System.exit(1);
    }
}

/**
* CS496 - Lab5 - XML-RPC Server/Client - Server Module
* @author: Marek Handl
* @date: February 22, 2007
* *****
* creates last lines of xml-rpc response
*/
public void finishResponse() {
    xml += " </params>\r\n" +
        " </methodResponse>\r\n";
}

/**

```

```

* CS496 - Lab5 - XML-RPC Server/Client - Server Module
* @author: Marek Handl
* @date: February 22, 2007
* *****
* Kilogram to pound conversion
*
* @param kg - kilogram
*/
private Object kgToLb( int kg ) {
    return ( (double)kg*2.20462262 );
}

/**
* CS496 - Lab5 - XML-RPC Server/Client - Server Module
* @author: Marek Handl
* @date: February 22, 2007
* *****
* Pound to kilogram conversion
*
* @param lb - pound
*/
private Object lbToKg( int lb ) {
    return ( (double)lb*0.45359237 );
}

/**
* CS496 - Lab5 - XML-RPC Server/Client - Server Module
* @author: Marek Handl
* @date: February 22, 2007
* *****
* Celsius to Fahrenheit conversion
*
* @param celsius
*/
private Object cToF( double celsius ) {
    return ( celsius * 9/5 + 32 );
}

/**
* CS496 - Lab5 - XML-RPC Server/Client - Server Module
* @author: Marek Handl
* @date: February 22, 2007
* *****
* Fahrenheit to Celsius conversion
*
* @param fahrenheit
*/
private Object fToC( double fahrenheit ) {
    return ( (fahrenheit - 32) * 5/9 );
}

/**
* CS496 - Lab5 - XML-RPC Server/Client - Server Module
* @author: Marek Handl
* @date: February 22, 2007
* *****
* Yard, foot, inch to meter conversion
*
* @param yard
* @param ft - foot
* @param inch
*/
private Object yfiToM( double yard, double ft, double inch ) {
    return ( yard*0.9144 + ft*0.3048 + inch*0.0254 );
}

/**
* CS496 - Lab5 - XML-RPC Server/Client - Server Module
* @author: Marek Handl
* @date: February 22, 2007

```

```

* *****
* Meter to yard, foot, inch conversion
*
* @param meter
*/
private Object mToyfi( double meter ) {
    int y = Double.valueOf(meter / 0.9144).intValue();
    int f = Double.valueOf( (meter - (double)y*0.9144) / 0.3048 ).intValue();
    double i = ( meter - (double)y*0.9144 - (double)f*0.3048 ) / 0.0254;
    String result = Integer.toString(y) + " yard " + Integer.toString( f ) + " foot " +
Double.toString(i) + " inch";
    return result;
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Server Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * Miles per hour to kilometers per hour conversion
 *
 * @param mph- miles per hour
 */
private Object mphTokmh( double mph ) {
    return ( mph * 1.609344 );
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Server Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * Kilometers per hour to miles per hour conversion
 *
 * @param kmh- kilometers per hour
 */
private Object kmhTomph( double kmh ) {
    return ( kmh / 1.609344 );
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Server Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * closes stream readers, writers and the socket
 */
private void closeConnection() throws IOException {
    out.close();
    in.close();
    socket.close(); // this ends the connection
    System.out.println( "Connection closed " + socket.getInetAddress() + ":" +
socket.getPort());
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Server Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * Parses string line and extracts a method argument if present.
 *
 * @param line - String, last line read from the incoming xml-rpc request
 * @return - Object, one of read arguments (int, double, boolean, string),
null in case the line did not include any argument
 */
private Object parseParameter( String line ) {
    if ( line.contains( "<i4>" ) ) {
        return Integer.parseInt( line.substring( line.indexOf( "<i4>" ) + 4,
line.indexOf( "</i4>" ) ) );
    }
}

```

```

    }
    else if ( line.contains( "<int>" ) ) {
        return Integer.parseInt( line.substring( line.indexOf( "<int>" )+ 5,
line.indexOf( "</int>" ) ) );
    }
    else if ( line.contains( "<double>" ) ) {
        return Double.parseDouble( line.substring( line.indexOf( "<double>" )+ 8,
line.indexOf( "</double>" ) ) );
    }
    else if ( line.contains( "<string>" ) ) {
        return line.substring( line.indexOf( "<string>" )+ 8, line.indexOf( "</string>"
) );
    }
    else if ( line.contains( "<boolean>" ) ) {
        return Boolean.parseBoolean( line.substring( line.indexOf( "<boolean>" )+ 9,
line.indexOf( "</boolean>" ) ) );
    }
    return null; // line does not include any known argument type
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Server Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * main function of the thread
 * opens stream readers and writers, parses user's request, calls appropriate methods and
sends result in a response
 */
public void run() {

    try {
        System.out.println( "\nClient connected: " + socket.getInetAddress() + ":" +
socket.getPort() );

        // open readers and writers for the socket
        out = new PrintWriter( socket.getOutputStream(), true ); // outgoing
messages (headers)
        in = new BufferedReader( new InputStreamReader( socket.getInputStream() ) );
// incoming messages (requests)

        String inputLine;
        while ( (inputLine=in.readLine()) == null ) {} // wait for buffer ready

        if ( debug ) System.out.println( inputLine );

        // **** parse request string ****

        // read command type
        if ( !inputLine.startsWith("POST ") ) { // wrong format of request
            System.out.println( "ERROR - wrong request format" );
            faultResponse( "Wrong request format" );
            return;
        }

        // read http type
        if ( inputLine.contains( "HTTP/1.0" ) ) {
            httpType = "HTTP/1.0";
        }
        else if ( inputLine.contains( "HTTP/1.1" ) ) {
            httpType = "HTTP/1.1";
        }
        else { // wrong http type
            System.out.println( "ERROR - wrong request format" );
            faultResponse( "Wrong http format" );
            return;
        }

        // read path
        int beginOfURL = inputLine.indexOf( ' ' ) + 1;

```

```

int endOfURL = inputLine.substring(beginOfURL).indexOf( ' ' ) + beginOfURL;
String URL = inputLine.substring( beginOfURL, endOfURL);
if ( debug ) System.out.println( "url: " + URL );
// if the path is not path to the rpc service, return fault message
if ( !URL.equals( rpcPath ) ) {
    System.out.println( "ERROR - wrong path" );
    faultResponse( "Wrong path - no RPC service" );
    return;
}

// *** read headers and do not display them ***
String line;
int length = -1;

while (true) {
    while ( !in.ready() ) {} // wait for buffer ready
    line = in.readLine();
    if ( debug ) System.out.println( line ); // print out headers
    if ( line.equals("") ) { // end of headers, data will start
        break;
    }
    if ( line.length() >= 15 && line.substring(0, 15).equals("Content-
Length:") ) { // length header is mandatory
        length = Integer.parseInt( line.substring(16).trim() );
    }
    // other headers are ignored
}

// *** read data and display them ***
// Length of data has been specified in the header
int state = 0;
String method = "";
Object param1 = null;
Object param2 = null;
Object param3 = null;
Object tmp;
if ( length != 0 ) {
    int readCharacters = 0;
    while ( readCharacters < length ) {
        while ( !in.ready() ) {} // wait for buffer ready
        line = in.readLine();

        if ( line == null ) { // no more data
            faultResponse( "Wrong length parameter!" );
            return;
        }

        // read method name and parameters
        switch (state) {
            // this switch could be enhanced to make a strict parser, but this
            way it works too and it tolerates some errors in syntax
            case 0: // reading method name
                if ( line.contains( "<methodName>" ) ) {
                    method = line.substring( line.indexOf( "<methodName>"
)+12, line.indexOf( "</methodName>" ) );
                    state++;
                }
                break;
            case 1: // reading parameter number 1
                tmp = parseParameter( line );
                if ( tmp != null ) {
                    param1 = tmp;
                    state++;
                }
                break;
            case 2: // reading parameter number 2
                tmp = parseParameter( line );
                if ( tmp != null ) {
                    param2 = tmp;
                    state++;
                }
        }
    }
}

```

```

        }
        break;
    case 3: // reading parameter number 3
        tmp = parseParameter( line );
        if ( tmp != null ) {
            param3 = tmp;
            state++;
        }
        break;
    default: // ignore any other lines
        break;
    }

    if ( debug ) System.out.println( line );
    readCharacters += line.length() + 2; // adding 2 because of
CRLF
    }
}
// length header is missing - just in case (length is mandatory for xml-rpc)
else {
    faultResponse( "Wrong length parameter!" );
    return;
}

// if parsing parameters failed, return fault response
if ( param1 == null ) {
    System.out.println( "No parameter" );
    faultResponse( "No parameters" );
    return;
}

// **** compute result and create response xml ****
createResponse();
Object result = null;
if ( method.equals( "kgToIb" ) )
    result = kgToIb( (Integer)param1 );
else if ( method.equals( "lbTokg" ) )
    result = lbTokg( (Integer)param1 );
else if ( method.equals( "cTof" ) )
    result = cTof( (Double)param1 );
else if ( method.equals( "fToc" ) )
    result = fToc( (Double)param1 );
else if ( method.equals( "yfiTom" ) && param2 != null && param3 != null )
    result = yfiTom( (Double)param1, (Double)param2, (Double)param3 );
else if ( method.equals( "mToyfi" ) )
    result = mToyfi( (Double)param1 );
else if ( method.equals( "mphTokmh" ) )
    result = mphTokmh( (Double)param1 );
else if ( method.equals( "kmhTomph" ) )
    result = kmhTomph( (Double)param1 );
else {
    faultResponse( "Unknown method or incorrect parameters!" );
    return;
}
if ( result == null ) { // in case computation failed
    faultResponse( "Unknown error - null result!" );
    return;
}

addParameter( result ); // encode result in xml-rpc format
finishResponse();

// **** send result back to the client ****
System.out.println( "Method: " + method + " ... input: " + param1.toString() +
" ... output: " + result.toString() );

String message = httpType + " 200 OK\r\n" +
"Content-Length: " + xml.length() + "\r\n" + headers +
"\r\n";

```

```

        message += xml;

        if ( debug ) System.out.println( "\n" + message );
        out.println( message );           // send response to the client

        // **** end connection and exit ****
        closeConnection();
    }
    catch (Exception e ) {
        System.out.println( "Something went wrong - server thread!!! " + e.getMessage()
);
    }
}
}
}

```

```

// ***** Client.java *****

```

```

/*
 * CS496 - Lab5 - XML-RPC Server/Client - Client Module
 * author: Marek Handl
 * date: February 22, 2007
 * *****
 * The project consists of two programs - a Server handling xml-rpc requests and a Client sending
the requests.
 *
 * This is the Client Module.
 * All input is done via command line parameters. There has to be at least three parameters.
 * First parameter - URL (host IP and path if necessary)
 * Second parameter - name of the method to be executed on the server
 * Third parameter - argument of the method
 * Additional parameters - in case the method takes more than one argument (method yfiTom)
 *
 * Program parses the command line parameters, puts them in a xml-rpc format, connects to the
server,
 * sends request, reads response and displays it to the screen. Fault responses are handled too.
 */

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;

/* Class Client is the only class in the Client program
 * See file documentation for functionality description
 */
public class Client {

    private static boolean debug = false;           // display detailed information? (used
for debugging usually)
    private static int port = 80;                   // port where the server listens
    private static String ip = "localhost";         // ip address of the server
    private static String path = "/RPC";           // path to the file
    private static String httpType = " HTTP/1.0"; // type of http protocol used for
communication
    private static Socket socket;                   // connection with the server
    private static PrintWriter out;                 // socket output
    private static BufferedReader in;               // socket input
    private static String xml = "";                // xml request is saved in this
variable

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Client Module
 * @author: Marek Handl

```

```

    * @date: February 22, 2007
    * *****
    * displays error message and ends the program
    */
public static void serverNotFound() {
    System.out.println( "ERROR - Server not found" );
    System.exit(1);
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Client Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * displays error message and ends the program
 */
public static void wrongURL() {
    System.out.println( "ERROR - wrong URL format" );
    System.exit(1);
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Client Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * closes stream-readers and closes socket
 */
public static void closeConnection() {
    try {
        in.close();
        out.close();
        socket.close();
    }
    catch (Exception e) {}
    System.exit(1);
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Client Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * it is called if there was a discrepancy in length of incoming data and length read from
header
 * it will read and display all incoming data regardless the length value, output will not be
formatted
 */
public static void lengthError() {
    if ( debug ) System.out.println( "MESSAGE ERROR - wrong length value" );
    String line;
    try {
        while ( (line = in.readLine()) != null ) {
            System.out.println( line );
        }
    } catch (Exception e) {}
    closeConnection();
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Client Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * creates first lines of xml part of the request
 *
 * @param methodName      - name of the method to be executed on the server
 */
public static void createRequest( String methodName ) {
    xml = "<?xml version=\"1.0\"?>\r\n" +

```

```

        "<methodCall>\r\n" +
            "<methodName>" + methodName + "</methodName>\r\n" +
            "<params>\r\n";
    }

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Client Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * encodes parameter into xml-rpc format
 *
 * @param parameter      - Object, (int, double, boolean or String) that carry the result
value
 */
public static void addParameter( Object parameter ) {
    // Integer
    if ( parameter.getClass().getName().equals( Integer.class.getName() ) ) {
        xml += "<param>\r\n" +
            "<value><i4>" + (Integer) parameter + "</i4></value>\r\n" +
            "</param>\r\n";
    }
    // Double
    else if ( parameter.getClass().getName().equals( Double.class.getName() ) ) {
        xml += "<param>\r\n" +
            "<value><double>" + (Double) parameter +
"</double></value>\r\n" +
            "</param>\r\n";
    }
    // String
    else if ( parameter.getClass().getName().equals( String.class.getName() ) ) {
        xml += "<param>\r\n" +
            "<value><string>" + (String) parameter +
"</string></value>\r\n" +
            "</param>\r\n";
    }
    // Boolean
    else if ( parameter.getClass().getName().equals( Boolean.class.getName() ) ) {
        xml += "<param>\r\n" +
            "<value><boolean>" + (Boolean) parameter +
"</boolean></value>\r\n" +
            "</param>\r\n";
    }
    // other types are not supported
    else {
        System.out.println( "ERROR - Unknown parameter type!" );
        System.exit(1);
    }
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Client Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * appends last lines to the request
 */
public static void finishRequest() {
    xml += "</params>\r\n" +
        "</methodCall>\r\n";
}

/**
 * CS496 - Lab5 - XML-RPC Server/Client - Client Module
 * @author: Marek Handl
 * @date: February 22, 2007
 * *****
 * main function of the Client
 * parses command parameters, connects to a server, sends request, reads response and
displays it

```

```

*
* @param args          - input, command line arguments
*/
public static void main(String[] args) {

    // **** parsing of command line parameters ****
    if ( args.length < 3 || args[0].equals("-h") ) {
        System.out.println( "ERROR - Wrong parameters" );
        System.out.println( "Usage: Client URL methodName parameter [parameter]");
        System.exit(1);
    }

    else {

        // ** parse the URL **
        String url = args[0];
        // remove http from the beginning of the url if present
        if ( url.toLowerCase().startsWith( "http://" ) )
            url = url.substring(7);
        // find where the host is in the url string
        int indexOfSlash = url.indexOf( '/' );
        if ( indexOfSlash == -1 ) { // just host, no path and filename
            indexOfSlash = url.length();
        }
        // parse port of the host if present
        int endOfHost = indexOfSlash;
        int indexOfColon = url.indexOf(':');
        if ( indexOfColon != -1 && indexOfColon < indexOfSlash ) {
            endOfHost = indexOfColon;
            try {
                port = Integer.parseInt(
url.substring(indexOfColon+1,indexOfSlash) );
            } catch (NumberFormatException e) {
                wrongURL();
            }
        }
        ip = url.substring( 0, endOfHost ); // just host without port (it may be a
DNS name instead of an IP address)
        //if ( indexOfSlash == url.length() ) // there was no path in the URL
        //path = "/";
        if ( indexOfSlash != url.length() )
            path = url.substring( indexOfSlash );

        // ** parse method name and parameters **

        String methodName = args[1];
        // kilograms to pounds
        if ( methodName.equalsIgnoreCase( "kgtolb" ) ) {
            createRequest( "kgTolb" );
            addParameter( Integer.parseInt( args[2].toString() ) );
        }
        // pounds to kilograms
        else if ( methodName.equalsIgnoreCase( "lbtokg" ) ) {
            createRequest( "lbTokg" );
            addParameter( Integer.parseInt( args[2].toString() ) );
        }
        // Celsius to Fahrenheit
        else if ( methodName.equalsIgnoreCase( "ctof" ) ) {
            createRequest( "cTof" );
            addParameter( Double.parseDouble( args[2].toString() ) );
        }
        // Fahrenheit to Celsius
        else if ( methodName.equalsIgnoreCase( "ftoc" ) ) {
            createRequest( "fToc" );
            addParameter( Double.parseDouble( args[2].toString() ) );
        }
        // yard, foot, inch to meter
        else if ( methodName.equalsIgnoreCase( "yfitom" ) ) {
            if ( args.length < 5 ) {
                System.out.println( "ERROR - yfiTom needs 3 parameters" );
            }
        }
    }
}

```

```

        System.exit(1);
    }
    createRequest( "yfiTom" );
    addParameter( Double.parseDouble( args[2].toString() ) );
    addParameter( Double.parseDouble( args[3].toString() ) );
    addParameter( Double.parseDouble( args[4].toString() ) );
}
// meter to yard, foot, inch
else if ( methodName.equalsIgnoreCase( "mtoyfi" ) ) {
    createRequest( "mToyfi" );
    addParameter( Double.parseDouble( args[2].toString() ) );
}
// miles per hour to kilometers per hour
else if ( methodName.equalsIgnoreCase( "mphtokmh" ) ) {
    createRequest( "mphTokmh" );
    addParameter( Double.parseDouble( args[2].toString() ) );
}
// kilometers per hour to miles per hour
else if ( methodName.equalsIgnoreCase( "kmhtomph" ) ) {
    createRequest( "kmhTomph" );
    addParameter( Double.parseDouble( args[2].toString() ) );
}
// unknown method
else {
    System.out.println( "ERROR - unknown method" );
    System.exit(1);
}
finishRequest();
}

// **** send request and read response ****

if ( debug ) System.out.println( "Trying to connect..." + ip + ":" + port );

try {

    // **** socket set-up ****
    socket = new Socket( ip, port ); // connect to server
    if ( debug ) System.out.println( "Connected to " + ip + ":" + socket.getPort()
+ "    local port: " + socket.getLocalPort() );
    out = new PrintWriter( socket.getOutputStream(), true ); // outgoing
messages
    in = new BufferedReader( new InputStreamReader( socket.getInputStream() ) );
// incoming messages

    // **** send an HTTP POST message to the server ***
    String message = "POST " + path + httpType + "\r\n" +
        "User-Agent: MareenClient/0.1\r\n" +
        "Host: " + ip + ":" + port + "\r\n" +
        "Content-Type: text/xml\r\n" +
        "Content-Length: " + xml.length() + "\r\n\r\n";
    message += xml; // add the xml encoded method name and parameters

    if ( debug ) System.out.println( message );

    out.print( message ); // send the request
    out.flush();

    // **** read the response from the server ****
    while ( !in.ready() ) {} // wait for buffer ready
    String line = in.readLine(); // first line of response - status info
    if ( debug ) System.out.println( line ); // display the status info
    if ( line.startsWith( httpType + " " + "404" ) ) {
        System.out.println( "ERROR - Page not found" );
    }
}

int length = 0;

// *** read headers and do not display them ***

```

```

while (true) {
    while ( !in.ready() ) {} // wait for buffer ready
    line = in.readLine();
    if ( debug ) System.out.println( line ); // print out headers
    if ( line.equals("") ) { // end of headers, data will start
        break;
    }
    if ( line.length() >= 15 && line.substring(0, 15).equals("Content-
Length:") ) { // length header is mandatory
        length = Integer.parseInt( line.substring(16).trim() );
    }
    // other headers are ignored
}

// *** read data and display them ***
// Length of data has been specified in the header
Object param = null;
int state = 0;
if ( length != 0 ) {
    int readCharacters = 0;
    while ( readCharacters < length ) {
        while ( !in.ready() ) {} // wait for buffer ready
        line = in.readLine();
        if ( line == null ) { // no more data
            lengthError();
        }
        switch (state) {
            case 0: // normal response
                if ( line.contains( "<fault>" ) ) {
                    state = 1;
                    break;
                }
                if ( line.contains( "<i4>" ) ) {
                    param = Integer.parseInt( line.substring(
line.indexOf( "<i4>" )+ 4, line.indexOf( "</i4>" ) ) );
                }
                else if ( line.contains( "<int>" ) ) {
                    param = Integer.parseInt( line.substring(
line.indexOf( "<int>" )+ 5, line.indexOf( "</int>" ) ) );
                }
                else if ( line.contains( "<double>" ) ) {
                    param = Double.parseDouble( line.substring(
line.indexOf( "<double>" )+ 8, line.indexOf( "</double>" ) ) );
                }
                else if ( line.contains( "<string>" ) ) {
                    param = line.substring( line.indexOf( "<string>" )+
8, line.indexOf( "</string>" ) );
                }
                else if ( line.contains( "<boolean>" ) ) {
                    param = Boolean.parseBoolean( line.substring(
line.indexOf( "<boolean>" )+ 9, line.indexOf( "</boolean>" ) ) );
                }
                break;
            case 1: // fault response - looking for faultString
                member
                if ( line.contains( "<name>faultString</name>" ) )
                    state = 2;
                break;
            case 2: // fault response - reading error message
                if ( line.contains( "<value><string>" ) ) {
                    param = line.substring( line.indexOf("<string>")+8,
line.indexOf("</string>" ) );
                }
                break;
            default:
                break;
        }
        if ( debug ) System.out.println( line );
        readCharacters += line.length() + 2; // adding because of CRLF
    }
}

```

```

    }
    // length header is missing - just in case (it is mandatory for xml-rpc)
    else {
        lengthError();
    }

    if ( param == null )        // no result has been parsed
        System.out.println( "No response data" );
    else if ( state >= 2 )     // there has been an error and fault response was

        System.out.println( "Failed: " + param.toString() );
    else // normal condition - result is valid
        System.out.println( "Response: " + param.toString() );

    // **** close connection and exit ****
    if ( debug ) System.out.println( "Closing..." );
    in.close();
    out.close();
    socket.close();
    if ( debug ) System.out.println( "Connection closed" );

} catch ( UnknownHostException e ) {
    serverNotFound();
}
catch ( Exception e ) {
    System.out.println( "ERROR - " + e.getMessage() );
}

}

}

```

➤ REMOTE PROCEDURES INTERFACES

double kgTolb(int kg)

- description: kilogram to pound conversion
- input: integer – kilogram value
- output: double – pound value

double lbTokg(int lb)

- description: pound to kilogram conversion
- input: integer – pound value
- output: double – kilogram value

double cTof(double celsius)

- description: Celsius to Fahrenheit conversion
- input: double – Celsius value
- output: double – Fahrenheit value

double fToc(double fahrenheit)

- description: Fahrenheit to Celsius conversion
- input: double – Fahrenheit value
- output: double – Celsius value

double yfiTom(double yard, double foot, double inch)

- description: yard, foot and inch to meter conversion
- input: three doubles – yard, foot, inch values
- output: double – meter value

String mToyfi(double meter)

- description: meter to yard, foot and inch conversion
- input: double – meter value
- output: String – text string including appropriate values (e.g. “1 yard 2 foot 3 inch”)

double mphTokmh(double mph)

- description: miles per hour to kilometers per hour conversion
- input: double – miles per hour value
- output: double – kilometers per hour value

double kmhTomph(double kmh)

- description: kilometers per hour to miles per hour conversion
- input: double – kilometers per hour value
- output: double – miles per hour value

➤ TESTING

Screenshot of the client screen – all possible methods were called.

```
C:\WINDOWS\system32\cmd.exe

D:\>java Client localhost KGtoLB 5
Response: 11.0231131

D:\>java Client localhost LBtoKG 11
Response: 4.9895160700000005

D:\>java Client localhost CtoF 23.15
Response: 73.67

D:\>java Client localhost FtoC 73.67
Response: 23.150000000000002

D:\>java Client localhost YFItoM 2 3 4
Response: 2.8447999999999998

D:\>java Client localhost MtoYFI 2.8448
Response: 3 yard 0 foot 4.000000000000014 inch

D:\>java Client localhost MPHtoKMH 75.4
Response: 121.34453760000002

D:\>java Client localhost KMHtoMPH 121.3445376
Response: 75.39999999999999

D:\>_
```

Screenshot of the server screen taken at the same time.

```
C:\WINDOWS\system32\cmd.exe - java Server

D:\>java Server
Server running.
Server IP address: MSOE-CNU6231RDM/155.92.107.81
Listening on port: 80
RPC path: /RPC

Client connected: /127.0.0.1:4688
Method: kgToLb ... input: 5 ... output: 11.0231131
Connection closed /127.0.0.1:4688

Client connected: /127.0.0.1:4689
Method: lbToKg ... input: 11 ... output: 4.9895160700000005
Connection closed /127.0.0.1:4689

Client connected: /127.0.0.1:4690
Method: cToF ... input: 23.15 ... output: 73.67
Connection closed /127.0.0.1:4690

Client connected: /127.0.0.1:4691
Method: fToC ... input: 73.67 ... output: 23.150000000000002
Connection closed /127.0.0.1:4691

Client connected: /127.0.0.1:4692
Method: yfiToM ... input: 2.0 ... output: 2.8447999999999998
Connection closed /127.0.0.1:4692

Client connected: /127.0.0.1:4693
Method: mToYfi ... input: 2.8448 ... output: 3 yard 0 foot 4.000000000000014 inc
h
Connection closed /127.0.0.1:4693

Client connected: /127.0.0.1:4694
Method: mphToKmh ... input: 75.4 ... output: 121.34453760000002
Connection closed /127.0.0.1:4694

Client connected: /127.0.0.1:4695
Method: kmhToMph ... input: 121.3445376 ... output: 75.39999999999999
Connection closed /127.0.0.1:4695
```

➤ **SAMPLE REQUEST**

Requesting conversion from Celsius to Fahrenheit degrees on localhost server. Input is 19.53 Celsius.

```
POST /RPC HTTP/1.0
User-Agent: MareenClient/0.1
Host: localhost:80
Content-Type: text/xml
Content-Length: 162

<?xml version="1.0"?>
<methodCall>
<methodName>cTof</methodName>
<params>
<param>
<value><double>19.54</double></value>
</param>
</params>
</methodCall>
```

➤ **SAMPLE RESPONSE**

Response received for the sample request.

```
HTTP/1.0 200 OK
Content-Length: 140
Connection: Close
Server: RPCServer/0.1
Content-Type: text/xml

<?xml version="1.0"?>
<methodResponse>
<params>
<param>
<value><double>67.172</double></value>
</param>
</params>
</methodResponse>
```