CS 360 Spring 2012   UPD Client and Server **Assignment One**
Client DUE: Feb 13, 2012, 2:15 pm **20 points**
Server DUE: Feb 17, 2012, 11:59pm **20 points**

For this project, you need to produce a UDP **client** that will interact with existing servers provided by the instructor and also write a UDP **server** to replace the instructor's server. The server is an online expression solver, the client can send a message containing a mathematical expression to the server and expect a response which is the evaluation of that expression. The version 1 of the protocol is defined below.

The expressions that can be solved by the server accept the following mathematical operators: +, -, /, and *. Operators are represented as ASCII characters. All operands are single-precision 32-bit IEEE 754 floating point numbers. Expressions are strictly evaluated from left to right, order of operations is not enforced and grouping symbols are not allowed. Expressions are allowed to have a maximum of 100 operators (101 operands). Responses containing the result of an expression have zero operators and a single operand which is the solution.

Each packet contains a code, an ASCII character, to denote what type of message that packet contains. A packet containing an expression for evaluation is marked with a 'Q', a response containing a solution is 'R', and a response message not containing a solution because of an error (divide by zero, mismatched operators, operands) is 'E'.

The first operand must begin on a 4 byte boundary. If the last row of operators does not fill a set of four bytes the unfilled bytes in that set are left unused as padding.

The packet layout is as follows (not to scale!):

```
<------- 4B ------->
+------------------+
|   Version        |
+------------------+
|opercount|CODE|PAD |
+---------+--------+
| op | op | PAD     |
| operand          |
: operand          :
| operand          |
+------------------+

Version: 0x1
opercount 0 - 100
CODE: 'Q': query, 'R': response, 'E': error (divide by zero, mismatched
operators/operands)
PAD: Reserved for future use, do not use.
op: '+', '-', '/', '*'
operand: single-precision 32-bit IEEE 754 floating point

For a response packet, opercount must be set to zero. Tthere must be no
operators or padding between the byte following the CODE and the
operand.
```

Your Server Must (Project Name: UDPMathPacket_Server_PUNetID):

Read the port number to listen on from the command line (args[0]).
Create a DatagramSocket on the given port.
Read a single DatagramPacket
Evaluate the packet
Produce a new DatagramPacket for a response.
Respond using the address:port provided in the received DatagramPacket
loop forever.

You may add debugging output to help. Your server must be single threaded.

Your Client Must be an Android App that will **(Project Name: UDPMathPacket_Client_PUNetID)**:

Read the address (coffee.cs.pacificu.edu) and port (12345) from the user.

Build a DatagramSocket
Allow the user to build an expression of at most 100 operators. This could be as simple as text box that you parse into the expression or some more structured UI for input.
(For a text box, you may require a space between each operand and operator and use **java.util.StringTokenizer** to parse the string in the text box into floats and operators. **Float.parseFloat(String s)** will produce a **float** from a **String**). An example UI is given at the end of this document.

Display the result, displaying an error if necessary.
Close the DatagramSocket

Optionally allow the user to re-enter the address and port
Optionally allow the user to submit a new expression.

You must also provide:

A short text document describing how to use your Android application.

Recommended schedule:

1. Build UI, Parse the expression, Build math packet.
2. Send math packet, watch via Wireshark to make sure it is formed correctly
(The CODE, and Ops are ASCII Text and therefore easily readable. Version and Opercount should also be easily readable. Operands are stored in single-precision 32-bit IEEE 754 floating point and are difficult to decode (but Google will help!)).
3. Send math packet, receive response, parse response, display.
4. Build server

To use Wireshark with an Android Emulator be sure to select the **any** network interface.
Be sure to try to connect your client to other students' servers as well as the instructor's server to help work out bugs.

Instructor's Servers:

The Java Version lives at: coffee.cs.pacificu.edu (64.59.233.248) port 12345
The C Version lives at: coffee.cs.pacificu.edu (64.59.233.248) port 12346
This machine is only available by name (coffee) from within the CS Lab. If you connect a device to the University's wireless network you must use the IP address.

Resources

```
http://docs.oracle.com/javase/tutorial/networking/datagrams/clientServer.html
Chapter 13, Java Network Programming
```

Project submission

Email me your Subversion repository and I will check out your two projects

Android Development

Build your Android project for the 1.6 API. This will allow you to run your project on any of the devices in the cabinet or the emulator. If you run your project on a device you **must** connect the device to the University's wireless network.   You should be able to use either BoxerSecure or BoxerGuest.

Demonstration

I expect a quick demonstration of your client on Feb 13 at the beginning of class.

Sharing code between projects

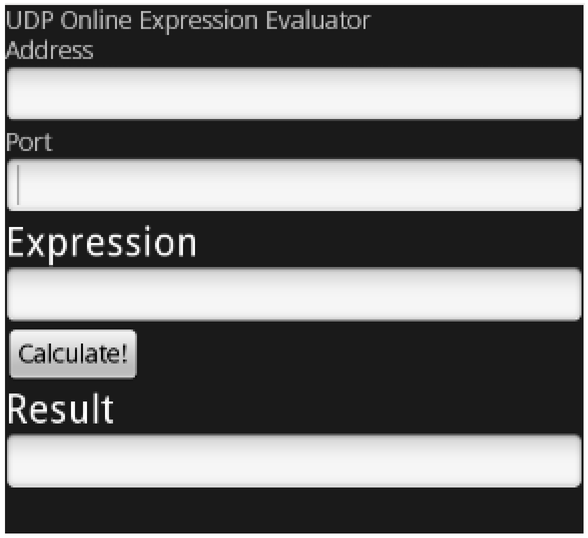
Both the client and server should have access to the **UDPMathPacket** class. There are two ways to do this. Put the UDPMathPacket class in one of the two projects. Have the other project depend on the project that contains the UDPMathPacket. (Right click project | Properties | Project References | select project). Import the UDPMathPacket, using the import statement, in the appropriate files in the project. Pay special attention to the package names.

Package Names:

All of your projects for this course should be in the package:
edu.pacificu.cs.cs360.PUNetID

All of your projects for this assignment should be in:
edu.pacificu.cs.cs360.PUNetID.UDPMath

In this UI, the user must fill out the Address, Port, and Expression text boxes before pressing Calculate! When the button is pressed, the text boxes will be read, the UDPMathPacket will be built and sent to the appropriate address:port. The response, either a value or error packet, will be displayed in Result. If an error packet is returned "Error!" is displayed.


UDP Online Expression Evaluator
Address
Port
Expression
Calculate!
Result