**Project M1: DNS server**  
  
You will develop a text mode DNS server. This server will listen on a user-specified socket for DNS lookup requests and return (to the client) a list of IP addresses with a preferred address. It must look and work **exactly** like this.  
  
SERVER VIEW (assume server runs on afsaccess1.njit.edu)  
  
afsaccess1-> ./dns 3344  
DNS Server Listening on socket 3344  
(1) Incoming client connection from [A.B.C.D:nnnnn] to me [E.F.G.H:nnnnn]  
    REQ: www.berkeley.edu  
  
(2) Incoming client connection from [A.B.C.D:nnnnn] to me [E.F.G.H:nnnnn]  
    REQ: www.stanford.edu  
  
(Note that A.B.C.D and E.F.G.H represent the client and server IP addresses and nnnnn is the ports.)  
  
CLIENT VIEW (assume client runs on afsaccess2.njit.edu)  
  
afsaccess2-> telnet afsaccess1.njit.edu 3344  
Trying 128.235.208.201 ...  
Connected to afsaccess1.njit.edu.  
Escape character is '^]'.  
www.berkeley.edu  
REQ www.berkeley.edu  
 IP = 54.149.224.140  
 IP = 54.149.224.140  
 IP = 52.39.79.24  
 IP = 52.39.79.24  
 IP = ::2600:1f14:436:7801:a429:76d7  
 IP = ::2600:1f14:436:7800:4a56:e1e2  
 PREFERRED IP = 54.149.224.140  
Connection closed by foreign host.  
afsaccess2-> telnet afsaccess1.njit.edu 3344  
Trying 128.235.208.201 ...  
Connected to afsaccess1.njit.edu.  
Escape character is '^]'.  
www.hogwarts.edu  
NO IP ADDRESSES FOUND  
Connection closed by foreign host.  
  
**REQUIREMENTS**  
  
You can develop this in C or in Java. Your code should have a comment block at the beginning which states the names and UCID (e.g. email address, not the student ID number) of the group members.

/\* Comment block  
   CS656-00N Group M1  
   Andy (ab123), Bianca (ba234), Charu (cv234), Dev (da3342)  
   Evelyn (es333), Farouk (fq989)  
\*/

Your program should have a main loop which will call a function (or Java method) called dns() which should perform the lookup and write the results into the client socket. Then you should close the connection. You may get bad input which should be handled as shown above. Your program should exit gracefully if there are errors and it must never crash or hang (that gets a zero!) BTW you can have more helper functions if you like but your code should not exceed 100 lines.  
  
Your dns() function **\*must\*** use getaddrinfo() in C OR InetAddress.GetAllByName(String) in Java to get the \*list\* of IP addresses. You should then design your own logic to select the preferred IP address.  
  
You must write the code yourself. You cannot use any other code (not even from the UNP book or the Oracle tutorials.) Any form of cheating attracts the maximum UAIP penalty i.e. zero on the project.  
  
*Some helpful hints*  
  
A good starting point is to study the code for the tcpserv01 in the UNP book and read the explanations. This will help you get a general structure for servers. Try to write a simple "echo" server on your own. Then add the dns() function etc. You should try to test the "parts" of your program individually.  
  
Important requirements for Java projects

* Java projects will be submitted as a SINGLE file (DNS.java) which will be compiled with the command line Java compiler (javac DNS.java) on AFS. We will run this as a standalone Java program from the command line. You can develop it wherever you like, but when you submit, it should be an independent Java class that can compile on its own, outside of an IDE or any additional libraries. If you have questions, please ASK.
* We will use the /afs/cad/linux/java8 Java system on AFS to compile AND run your code.
* You must import ALL Java packages/classes explicitly e.g. java.io.InputStream. You cannot "import java.whatever.\*"
* You cannot use String or StringBuffer or Character or any of the "higher order" data wrapper classes. The only exception is to use String in a constructor (as an unnamed variable) to pass to another constructor. There is very limited latitude on this, so ASK if you have questions. Using un-permitted higher order classes will result in penalties!
* You must use the primitive byte[] or char[] types for all data buffers and during the parsing of the request. (Yes, this means you'll have to deal with the charset/s during parsing. Such is life - you'll have to do this in your job anyway. And yes, you'll have to deal with byte[] for binary data anyway, so you won't gain much by wrapping it up in a String.)
* You must use ServerSocket and Socket but no other higher/lower order Socket classes are permitted. (SocketImpl is not permitted directly.)
* You must use ServerSocket.bind(InetAddress,int). You cannot accept() directly, and you cannot use any other ServerSocket methods that directly listen() and then accept().
* All exceptions MUST be caught at their lowest level and a printStackTrace() to System.out must be called. I will be studying your code very carefully so as to make it equitable with C - to demonstrate that you've understood the various socket exception conditions. Specifically, exception handlers that smother exceptions will be run without the exception handling, so be careful when testing. Writing try { ... } catch (Exception) is an example of a blanket smothering exception.

**SUBMISSION**  
You will submit a single C file (dns.c) or Java file (DNS.java) which we will compile using gcc -Wall (there should be no warnings) or javac.

**GRADING**

This project is worth 10% of your class grade. 2 points are for a comment block and for compiles without warnings. Crash/hang = zero.