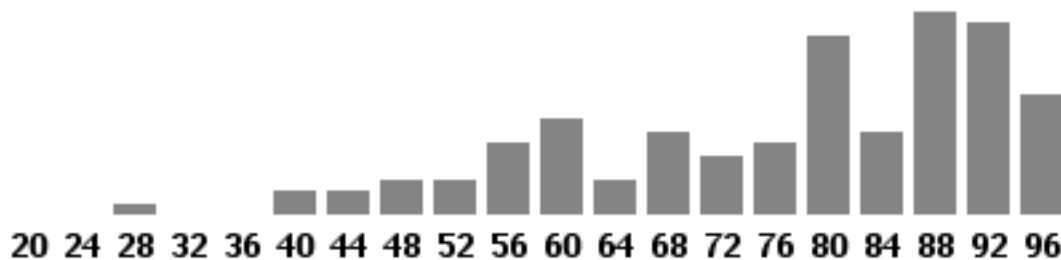


CS193i Exam+Solution

I thought the exam was pretty challenging. The scores do not form a gaussian distribution. Instead, it's just a sort of smear of scores that trails off into the 40's. Perhaps this reflects the wide range of students who take CS193i. (mean = 78, median=82)



1. Socket (10)

For this problem, you will write "delivery" socket Perl code that picks up text from one host and delivers it to another. You may assume that all text lines use a simple "\n" end-line convention. Ignore all errors -- assume that all I/O operations succeed and that all the data is formatted correctly. To connect client sockets, assume that a Perl "conn" function is defined:
`conn(SOCK, host-name, port-num).`

- Open the file "hosts". Each line in the file contains a hostname and port number separated by a space. We'll call these the 'X' hosts...

```
aaa.foo.com 8080  
bbb.bar.com 9090  
...
```

- Make a socket connection to each X host on the given port number. It will send back a sequence of text lines, followed by a blank line.
- After the blank line, the X host will send one or more lines that identify hosts and port numbers with the same syntax as above. We'll call these the 'Y' hosts. Open a connection to each Y host, send it the text lines from the X host from before the blank line, and close the connection.

```

blah blah blah
yatta yatta
text text text
<blank line>
ccc.abc.com 123
www.bbb.edu 1000

```

Your Perl code here...

```

#!/usr/bin/perl
# Open "hosts" file
open(F, "hosts");

# Get and use host information from "hosts"
while ($x = <F>) {

    # Parse information associated with X
    $x =~ m/(.+)\s(\d+)/;
    $xhost = $1;
    $xport = $2;

    # Connect to X
    conn(X, $xhost, $xport);

    # Read text from X
    $text = "";
    while (($line = <X>) ne "\n") {
        $text .= $line;
    }

    # Get and use host information from X
    while ($y = <X>) {

        # Parse information associated with Y
        $y =~ m/(.+)\s(\d+)/;
        $yhost = $1;
        $yport = $2;

        # Connect to Y and send Y text
        conn(Y, $yhost, $yport);
        print Y $text;
        close(Y);
    }

    close(X);
}

close(F);

```

Basic Criteria:

- * Solution should open "hosts" with call to open function (1 point).
- * Solution should correctly iterate through the lines in "hosts", and connect to each host (1 point). Doing so requires parsing each line in "hosts" using either an appropriate regular expression, or an appropriate call to split (1 point).
- * Solution should correctly accumulate text from X in a local

variable (2 points).

- * Solution should correctly detect the blank line sent by X. Doing so requires making a comparison to "\n" (2 points).
- * Solution should correctly iterate through each host, Y, sent by X (1 point), connect to each host, Y, (which, again, requires appropriate parsing) (1 point), and send the text from X to the each host, Y (1 point).

Miscellaneous Criteria:

- * Solution should use the correct EOL ("\n" rather than "\r\n") (1 point deduction).
- * Solution should pay attention to the fact that X hosts send text information before they send host information (1 point deduction).
- * Solution should not involve HTTP GET requests (2 point deduction).
- * Solution should not interpret the blank as a host, Y (1 point deduction).

2. HTTP (10)

For this problem you will write some simple Java HTTP client code. You may omit all error handling and exception code. Complete the code for the method

```
public String webSearch(String host, String path, String target) {
```

- The host and path parameters represent the data from a url. For the url "http://foo.com/bar", the host parameter is "foo.com" and the path parameter is "/bar".
- Connect to port 80 on the given host (code provided below)
- Send an HTTP request (1.0 or 1.1) for the given path.
- Read back the HTTP response. (You may ignore the "200" result code -- proceed regardless or the result code)
- If the returned HTTP document body is a textual and it contains the target string somewhere in one of its lines (case-sensitive), then return the first matched line, otherwise return null.

```
/* Solution code for Question 2 */
```

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

```
public class WebSearch {
```

```
    public String webSearch(String host, String path, String target) {
```

```
        try {
            Socket sock = new Socket(host, 80);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(sock.getInputStream()));
```

```
            PrintWriter out = new PrintWriter(sock.getOutputStream());
```

```
            /* If chose to do HTTP/1.1, had to also include
             * a host header. The GET, path, and having two
             * newlines (a blank line after the request) were
```

```

    * all worth a point.
    * 3 points
    */
    out.print("GET " + path + " HTTP/1.0\r\n");
    out.print("\r\n");
    out.flush();        // (did not grade off for this)

    String line;
    boolean inBody = false;

    while((line = in.readLine()) != null) {
    /* Search for content-type header, then
    * if there is not a text in that line, we
    * need to return null.
    * Iterating through headers, finding content-type,
    * and returning null were all worth one point.
    * 3 points
    */
    if (!inBody &&
        line.toLowerCase().indexOf("content-type") != -1) {
        if (line.toLowerCase().indexOf("text") == -1) {
            return null;
        }
    /* Correctly finding the start of the body is
    * worth a point.
    * 1 point
    */
    } else if (line.equals("")) {
        inBody = true;
    }

    /* Need to be in body to look for target,
    * which was worth a point. Returning the line
    * or null was worth two points.
    * 3 points
    */
    if(inBody && line.indexOf(target) != -1) {
        return line;
    }
    }
    return null;
} catch (Exception e) {
    return null;
}
}
}
}

```

3. CGI (30)

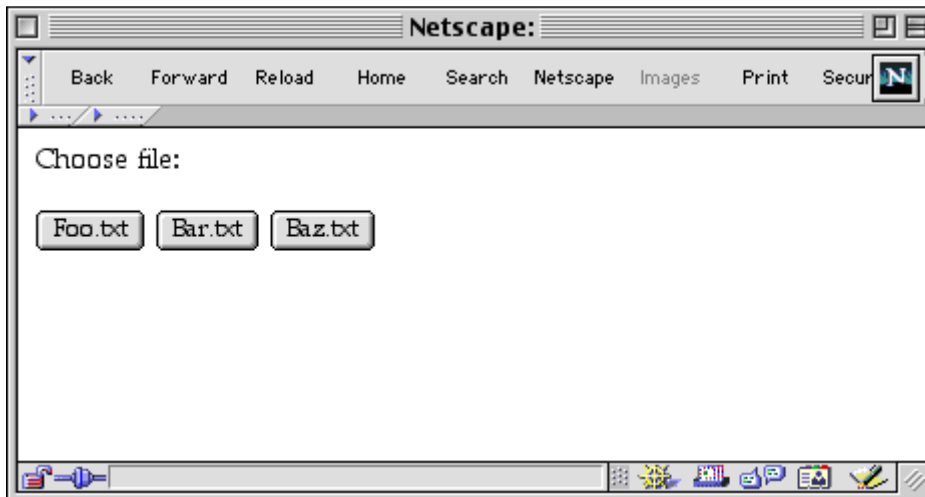
For this problem, you will implement a 3-page CGI that lets people edit their files on the CGI server. Here is the first page (P1), used to get the user name from the client side...



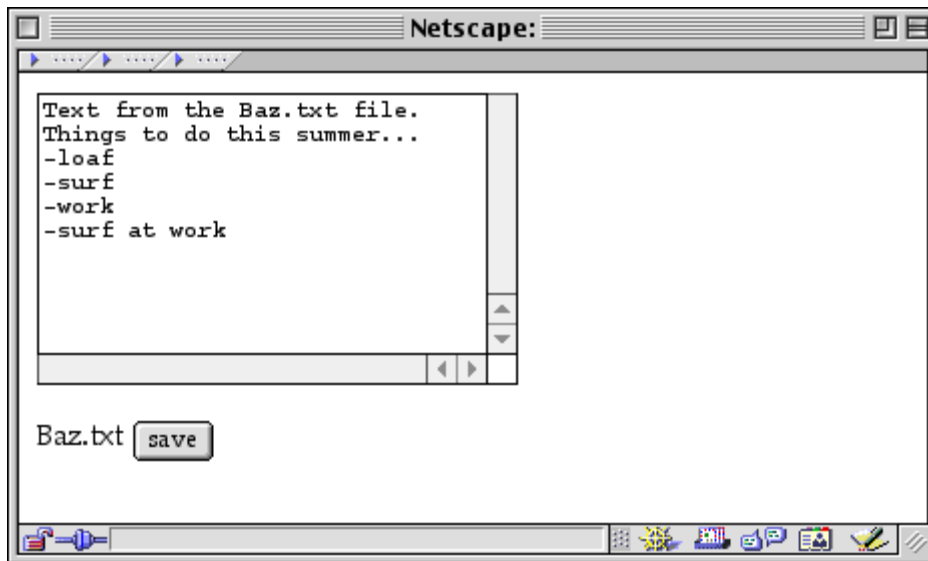
On the server side, there is a "users" text file. Each line in the file gives a username followed by the files they can edit, each separated by a space. The username and the filenames do not contain spaces.

```
bob Foo.txt resume.txt binky.txt
alice jokes.txt Bar.txt
....
```

If the user submits P1 with a valid username they get P2 (below), otherwise they get P1 again. P2 lets the user choose from among the files they can edit...



Choosing a file on P2 leads to P3, where the user can see and edit the text contents from the given file (assume that the file contains plain text).



The "save" button saves the text back to the file, and returns to the P2 for this user. (The "text from the Baz...." is the first line in the file in this case.)

Assumptions

- Ignore all error cases and do not worry about security.
- Do not worry about the back button -- the user must move forward.
- The Perl expression `split($str)` returns an array of the strings in `$str` that were separated by whitespace.
- Use formss to implement the solution, not cookies. (The same as HW3).

```
## This is the largest and most difficult problem on the exam.
#!/usr/bin/perl

use CGI;

$q = new CGI;

##### Main logic #####

print "Content-type: text/html\015\012\015\012";
print "<html><body>";

my $action = $q->param("action");

if ( !defined($action) ) { # first visit, so show the main page
    ShowMainPage();
} else {
```

```

# global user database
@userDB = ReadUserDB();

if ($action eq "login") {
    ShowChooseFilePageFor( $q->param("username") );
} elseif ($action eq "edit") {
    ShowEditPageFor( $q->param("username"), $q->param("filename") );
} elseif ($action eq "save") {
    WriteFile( $q->param("filename"), $q->param("filetext") );
    ShowChooseFilePageFor( $q->param("username") );
} else { # unknown action. just show main page
    ShowMainPage();
}
}

print "</body></html>";

##### helper functions #####

sub ShowMainPage {
    print "<form>";
    print "User: <input type=text name=username> ";
    print "<input type=submit name=action value=login>";
    print "</form>";
}

sub ShowChooseFilePageFor {
    my ($username) = @_;

    @filenames = GetUserFilenames($username);

    if ( !defined(@filenames) ) { #user not found in database
        ShowMainPage();
    } else {
        print "Choose file:<p>";
        print "<form>";

        # print a button for each filename
        foreach $file (@filenames) {
            print "<input type=submit name=filename value=$file> ";
        }

        # the username is propagated in this hidden field
        print "<input type=hidden name=username value=$username>";

        # the edit action is indicated in this hidden field
        print "<input type=hidden name=action value=edit>";
        print "</form>"
    }
}
}

```

```

sub ShowEditPageFor {
    my ($username, $filename) = @_;

    @filetext = ReadFile($filename);

    # show the textarea with existing file text
    print "<form><textarea name=filetext>";
    foreach $line (@filetext) {
        print $line;
    }
    print "</textarea><p>";

    # propagate username and filename in hidden fields
    print "<input type=hidden name=username value=$username>";
    print "<input type=hidden name=filename value=$filename>";

    print "$filename <input type=submit name=action value=save>";
}

# read and return user database
sub ReadUserDB {
    open(DB, "users");
    @allLines = <DB>;
    close(DB);

    return @allLines;
}

# returns an array of filenames belonging to a user
# or undef if the user was not found in the database
sub GetUserFilenames {
    my ($username) = @_;

    foreach $line (@userDB) {
        if ($line =~ /^$username\s/) {
            @filenames = split(' ', $line);
            shift (@filenames); #remove the username
            return @filenames;
        }
    }

    # user not found
    return undef;
}

# returns the array of lines in the file
sub ReadFile {
    my ($filename) = @_;

    open(FILE, $filename);
    @lines = <FILE>;
    close(FILE);

    return @lines;
}

```

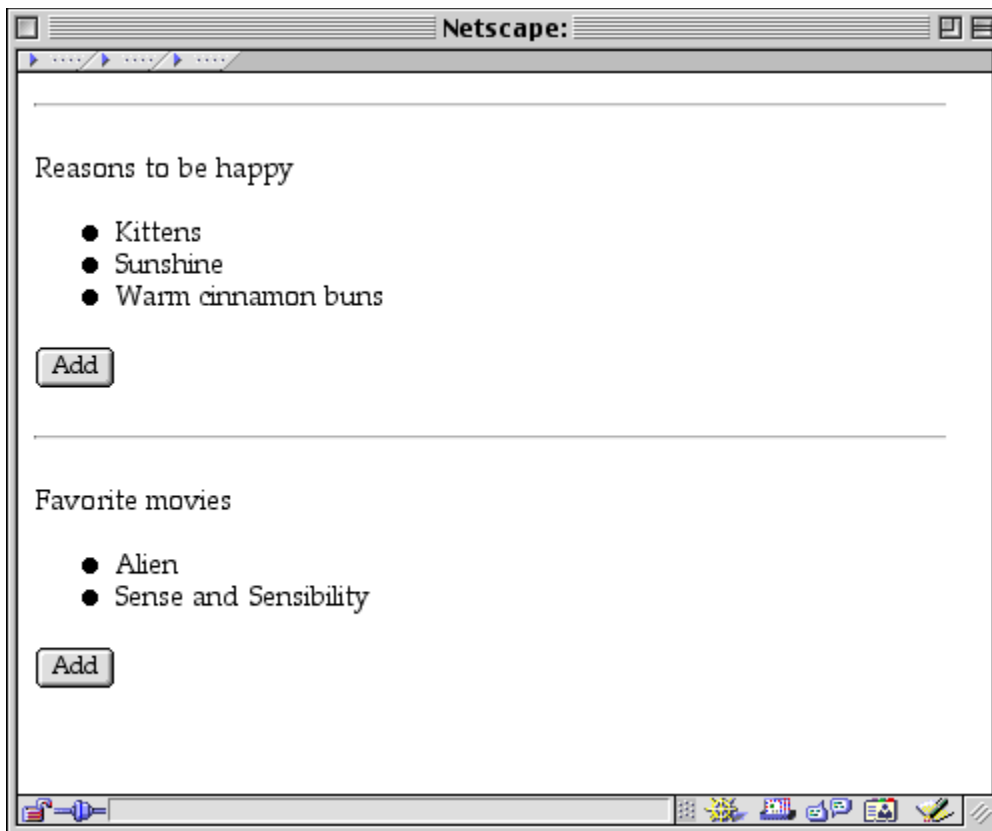
```
# writes the data to a file
sub WriteFile {
    my ($filename, $filetext) = @_;

    open(FILE, ">$filename");
    print FILE $filetext;
    close(FILE);
}
```

4. Servlet (15)

For this problem, you will implement a Java Servlet that implements a trivial sort of weblog that presents categories where people can add their comments (slashdot.edu?). The basic features are simple, except we add the requirement that the servlet resist user errors with the browser Back button (detailed below).

Here is the first page (P1) the user sees. There are several categories ("Reasons to be happy", "Favorite movies"), and within each category, users can add their comments for all to see...

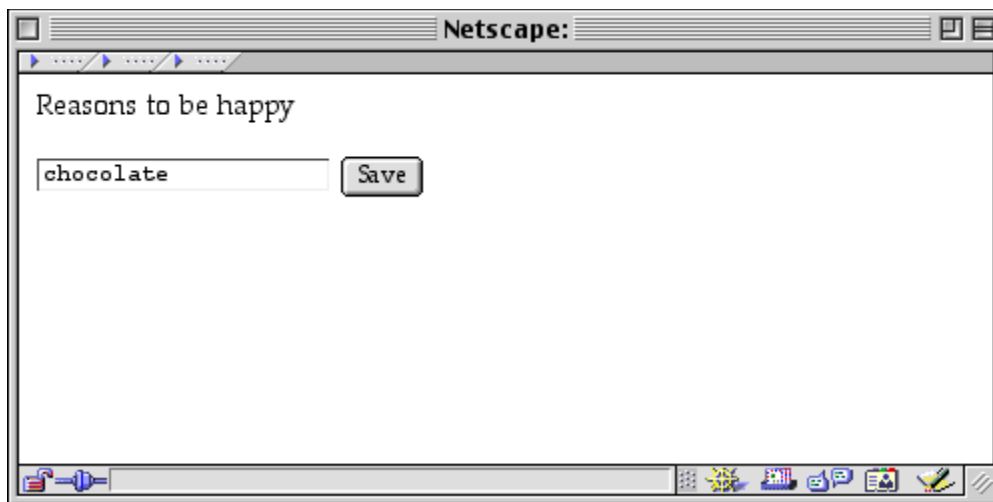


In the Servlet, the data is stored in a 'categories' instance variable that is an array of Category objects. Assume that the Category class is already defined, and responds to these messages...

```
public class Category {
    public String getTitle();           // get the title
    public String[] getComments();     // get the array of comments
    public void addComment(String comment); // add a comment
    ...
}
```

Your code can simply use the 'categories' array and the Category objects to access the data; Assume that the Category class deals with the back-end file or database storage. The number of categories does not change.

When the user clicks the Add button, they see page 2 (P2) for that category. It shows the category title and has a form for adding a comment...

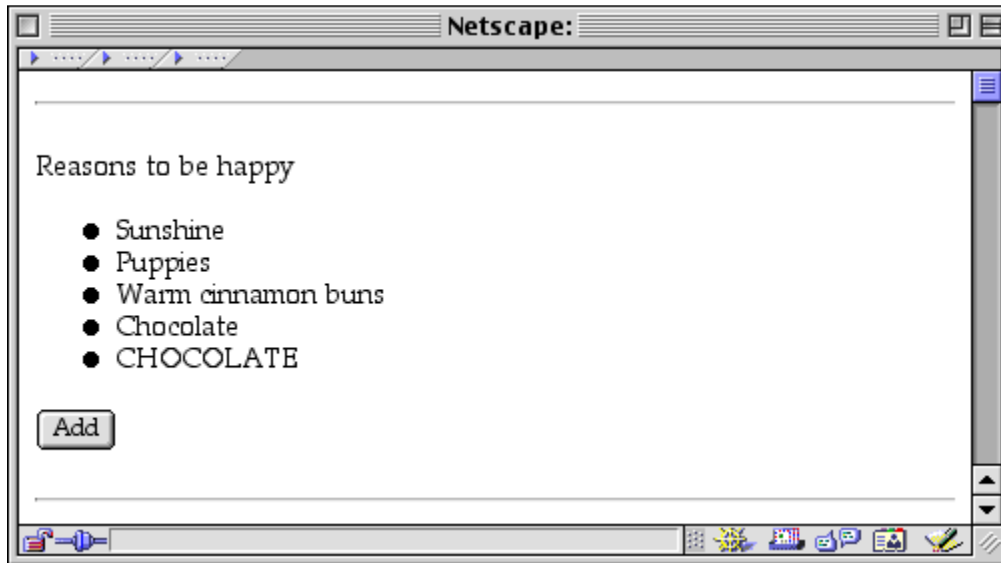


Clicking the Save button adds the comment into the data for that category, and displays P1.

On top of this fairly simple structure, we will add one tricky requirement. The servlet must resist a specific user error due to the browser Back button....

- Suppose the user goes to P2, types in the comment "Chocolate", and clicks the Save button. This adds the comment and displays P1.
- Then the user clicks the browser Back button, to re-visit P2, changes the comment to "CHOCOLATE", and then clicks the Save button.

- **Problem:** with the obvious Servlet (or CGI) implementation, this sequence will cause two comments to be added, which is probably not what the user wanted. In fact the same problem occurs if the user just clicks the browser Reload button when viewing P1 after a save



Your servlet must be resistant to the following specific "double submit" scenario:

Suppose the user clicks the browser Back button some number of times so they see an old instance of P2, and then click Save. Such a Save should be recognized as a "double submit". In that case, the servlet should not add a new comment or change the data in any way; it should just display P1. Even if the user changes the comment string in some way, the double-submit should be detected.

To implement this feature, the servlet should put some identifying information in each P2 when it is generated, to detect if it is ever submitted a second time.

You may ignore servlet threading issues (we never covered that detail). Do not use cookies explicitly, but everything else (forms, ivars, sessions, ...) is ok.

```
// Problem 4 Solution
// Basic idea: give each p2 a unique id. In the session,
// store the id's that have already be submitted, and notice
// if one is submitted a 2nd time.
//
// The point break up for this problem is:
// 5 points for getting just the web-form related functionality (
// P1->P2->P1...) and users can save comments.
// 10 points for putting in a guard to the double submit
// problem
//
// Most people got the 5 points for the web-form. As for the
// double submit portion, the most elegant solution (10 points)
```

```

// involves storing a table of IDs in the HTTP session. The IDs
// are monotonically increasing. When the session is initialized,
// we set the nextId to be assigned to be 0, and the list of seenIds
// to be empty. In P2 (the add page) we put the "nextId" in the form
// and increment it in the session. Then when the user submits the
// form the ID is sent to the servlet since this is the first time
// this ID is sent it would not be in the ID table and the save
// would be successful. However if the user hits back and resubmit,
// or reload, the ID would be present in the ID table and the save
// would not take effect.
//
// A very close but slightly less elegant solution (9 points) is to place some
// sort of ID in P2, and assume that when the "Save" code path of
// the servlet is invoked, that same ID will be in the HttpRequest.
// This would work if the user only works within a single browser.
// However, had he had two different browser windows and were editing two
// separate comments in the two windows, only the later comment would
// be allowed to be saved to the server. There are also problems with
// one browser window for certain valid back/forward sequences.
//
// An even less elegant solution (7 points) involves storing a history list as
// an ivar of the Servlet class, or using some sort of a global "log file"
// keep track of all the session IDs. This works, but does not take advantage
// of the "session" notion to store (and discard) the data for just one user.
//
// Some even less elegant solutions (5 points) could be using a global ID
// and expect that server will only issue an ID in "Add" and see it immediately
// sent back in "Save." However, when multiple users are adding comments,
// one user could easily disallow the add action of another user.
//
// Finally, comparing the content of a comment to be added with the
// comments already present is not good enough (5 points). Different users
// could add comments that are identical, and the same user could
// hit "Back" and modify the comment and resubmit. Both cases
// would cause undesirable behavior.

public class Weblog extends HttpServlet {
    private Category[] categories;

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException {

        String action = request.getParameter("Submit");
        if (action == null) {
            // P1 served after if-chain
        } else if (action.equals("Save")) {
            saveComment(request);
        } else if (action.equals("Add")) {
            serveP2();
            return;
        }

        serveP1();
    }

    private void initSession(HttpSession s) {
        seenIds = new Hashmap(); // (or could use ArrayList)
    }
}

```

```

// Stores IDs that the session has seen already; note that
// One could skip a table and just always compare the
// the ID in the save form against the last one given out,
// However that will not work for some valid uses of the
// backward/forward buttons.
nextId = new Integer(0);
s.setAttribute("seenIds", seenIds);
s.setAttribute("nextId", nextId);
}

private int assignId(HttpServletRequest request) {
    HttpSession s = request.getSession(true);
    if (s.isNew()) {
        initSession(s);
    }
    nextId = (Integer)s.getAttribute("nextId");
    int id = nextId.intValue();
    s.setAttribute("nextId", new Integer(id + 1));
    return id;
}

private void serveP1(PrintWriter out) {
    out.println("<html><body>");
    for (int i=0; i<categories.length; i++) {
        Category c = categories[i];
        out.println("<hr><p>" + c.getTitle() + "</p>\n<ul>");
        String [] comms = c.getComments();
        for (int j=0; j<comms.length; i++) {
            out.println("\t<li>" + comms[j] + "</li>");
        }
        out.println("</u>");
    }
    out.println("<form><p><input type=submit name=Submit
value=Add></p></form>");
    out.println("</body></html>");
}

private void serveP2(PrintWriter out, int catId) {
    out.println("<html><body>");
    out.println("<p>Reasons to be happy</p>");
    out.println("<form><p><input type=text name=NewComment>");
    out.println("<input type=submit name=Submit value=Save></p>");
    out.println("<input type=hidden name=CatId value=" + catId + ">");
    out.println("<input type=hidden name=CommentId value=" + assignId() + ">");
    out.println("</form>");
    out.println("</body></html>");
}

private HashMap getSeenId() {
    HttpSession s = request.getSession(true);
    if (s.isNew()) {
        initSession(s);
    }
    HashMap seenIds = (HashMap)s.getAttribute("seenIds");
    return seenIds;
}

private void saveComment(HttpServletRequest request) {

```

```

int catId;
Integer commentId;
String comment;

if ( (comment = request.getParameter("NewComment")) == null) {
    // Cannot save; comment not found
    return;
}

try {
    catId = Integer.parseInt(request.getParameter("CatId"));
    commentId = new Integer(request.getParameter("CommentId"));
} catch (Exception e) {
    // Cannot save; unable to determine catId or commentId
    return;
}

if ( (comment = request.getParameter("NewComment")) == null) {
    return;
}

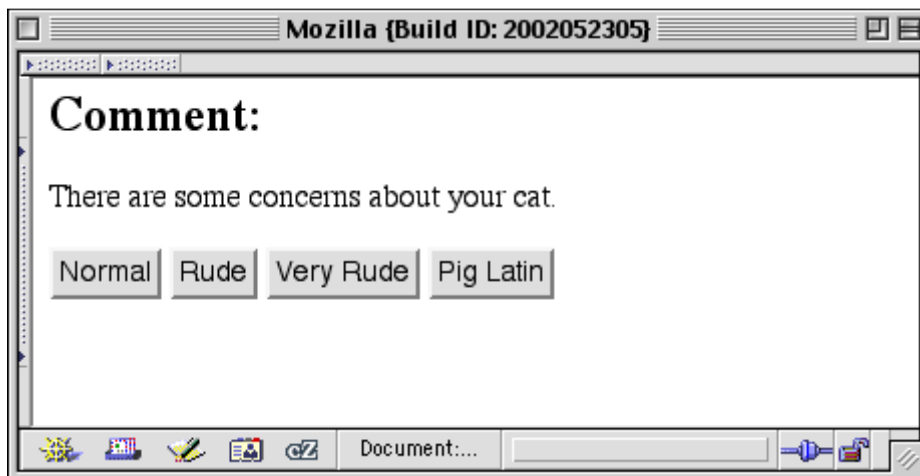
HashMap seenIds = getSeenId();
if (seenIds.put(commentId, commentId) != null) {
    // cannot save; had a duplicate in the seenIds
    return;
}

// Need to save
categories[catId].addComment(comment);
}
}

```

5. JavaScript / Thick Client (15)

Suppose you are working on a servlet, and part of its output is a little "comments" section like this...



The servlet has different style versions of the same comment

<u>Style</u>	<u>Comment</u>
Normal	There are some concerns about your cat.
Rude	Your cat sucks.
Very Rude	Your cat sucks, and so do you.
...	

Suppose in the servlet, you have two arrays of identical length. The "comments" array contains the various style versions of the comment string, and the "style" array identifies the corresponding style ("Normal", "Rude", ...). The exact length and contents of the arrays vary at runtime.

Objective: When first displayed, the page should use the first comment in the array. The user should be able to click a button to change the comment style. For example, clicking the "Rude" button should change the comment to "Your cat sucks". In typical JavaScript fashion, this should happen without hitting the server. (There are quite a few workable JavaScript strategies to accomplish this -- anything that works is fine.)

Write the necessary fragment of servlet code in the `doGet()`, below, to produce the Comments section, and its buttons and JavaScript code. You may write part of your solution as a JSP if you wish. You do not have to write out the `RequestDispatcher.....include(...)` fragment. We'll assume it's there if needed. Assume that the variables `out`, `comments` and `styles` are defined...

```

/* Solution code for Question 5 */
/*
Basic idea:
-have a JavaScript array with the comments in it
-have a JavaScript function that, given an int,
sets the HTML to the correct text from the array
-have a button for each style that calls the above
function. It's key to pass some sort of argument to the
function so it knows which comment to use.

Alternative: don't have an array. Instead, code the buttons
to pass the desired text directly to the function.
*/

import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ThickClient extends HttpServlet {

    String[] comments = {"There are some concerns about your cat.",
        "Your cat sucks.",
        "Your cat sucks, and so do you."};
    String[] styles = {"Normal", "Rude", "Very Rude"};

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {

```

```

response.setContentType("text/html");

    PrintWriter out = response.getWriter();

out.println("<HTML>\n<HEAD>\n<TITLE>Thick Client</TITLE>");
out.println("<SCRIPT LANGUAGE=\"JavaScript\">");
out.println("    var comments = new Array();");
for (int i = 0; i < comments.length; i++) {
    out.println("    comments.push(\"" + comments[i] + "\");");
}

out.println("");

out.println("    function setText(id, contents) {");
out.println("        var node = document.getElementById(id);");
out.println("        if(node != null) node.innerHTML = contents;");
out.println("    }\n");

out.println("    function generate(index) {");
out.println("        setText(\"comment\", comments[index]);");
out.println("    }\n");

out.println("</SCRIPT>\n<BODY>");
out.println("<H1>Comment:</H1><BR>");
out.println("<DIV id=\"comment\">" + comments[0] + "</div><BR>");

out.println("<TABLE>\n<TR>");
for(int i = 0; i < comments.length; i++) {
    out.println("    <TD><BUTTON onClick=\"generate("
        + i + ");\">" + styles[i] + "</BUTTON></TD>");
}
out.println("</TR>\n</TABLE>");

out.println("</HTML>");
}

/*
* Point Values:
* Getting Data from Servlet to JavaScript (4 points)
*     - Declare array (1 point)
*     - Create arrays of comments (3 points)
*     Easiest way was to follow examples and push comments
*     onto array. Can also do brief solution which does
*     setting from within button's onClick, or can do
*     call JavaScript function with comment as parameter.
*
* Buttons (9 points)
*     - Title and value (1 point)
*     - Choosing correct comment (5 points)
*     Need to have way to uniquely determine which button was
*     pushed. This necessitates passing a parameter to the
*     JavaScript function.
*     - Sending data to HTML (3 points)
*     Should actually place comment within HTML, not in a text
*     field. Also need to place at a particular location,
*     which is very easy with a <div id="x"> tag.
*
* DIV Tag (2 points)

```

```

    *      - Exists and has ID (2 points)
    */
}

```

6. Short Answer (20)

We are looking for **very short** answers for these. You do not need to justify your answers. Correct = +points, Blank = 0. Wrong = -points. Leave the question blank if you are not sure. You cannot get fewer than zero points for this question.

Grading: each part is worth either 1 or 2 points. Blank is worth 0 points, and a wrong answer is worth -1 or -2 points. If the answer given was not wrong, but was not specific enough, then it was just a 0 instead of negative.

a. What is the "referrer" field in an HTTP request?

The URL of the page that contained the URL that resulted in this request (1)

b. For an HTTP redirect, there are two key lines in the HTTP response. Give an example of each line.

Two parts

HTTP/1.0 301 Moved (1)

*Location: http://foo.com/bar.html (1)
(must have an absolute URL, since relative URLs are not technically correct)*

c. True or false: In HTTP, when using chunked transfer encoding, the content-length must be declared in the HTTP response header.

False (2)

The whole point of chunked transfer encoding is that the content-length field may be omitted

d. Suppose a client makes a request for the url "http://foo.com/a/b", but "b" is actually a directory. What is the most likely server response?

This will result in a 3xx redirect with the trailing slash: http://foo.com/a/b/ (2)

e. True or false: HTTP persistent connections allow the client to make multiple requests on a socket.

True (1)

f. True or false: HTTP cookie values are set on the HTTP response, and the values are returned on the HTTP request.

True (2)

Suppose Alice wants to do secure communication with Bob, but neither has a certificate, so they are vulnerable to an MITM attack. Alice sends the Alice.pub key to Bob so that he can securely send back a session key, k. Unfortunately, Carl is performing an MITM attack on all their communication...

g(1). What is the first thing that Bob receives?

Carl's public key, aka Carl.pub (2)

g(2). Bob sends back a session key, Bob.k. What does Alice receive? (there are a couple possibilities -- identify one)

Alice receives a session key, either Bob.k or some key made up by Carl. The key is encoded under Alice.pub (+1 bonus point for mentioning the Alice.pub layer of encryption) (1)

h. True or false: If the network administration tool for a machine reports that 'promiscuous mode is off', then we can conclude that the machine is not sniffing the network traffic of its neighbors.

False, see the "rootkit" lecture (2)

i. Suppose that Bob has keys Bob.pub and Bob.priv, and Alice wishes to send a document to Bob securely. True or false: Alice encrypts the document with Bob.pub, sends the encrypted form to Bob, and he decrypts with Bob.pub.

False, decrypts with the Bob.priv (1)

j. True or false: A "warhol" worm spreads quickly because its inner "probe" loop is coded to send packets very quickly.

False, A warhol worm optimizes the selection of hosts to probe (2)

k. Suppose a NAT router has the IP address 1.2.3.4 on its WAN/Internet side, 192.168.1.1 on its LAN side, and there is a host on its LAN side with address 192.168.1.100. That host wants to communicate with 9.8.7.6 out on the Internet, and so sends out a packet of the form (From:192.168.1.100 To:9.8.7.6). True or false: The NAT router rewrites the outgoing packet to the form (From: 192.168.1.1 To: 9.8.7.6).

False. it's from: 1.2.3.4 (2)