HW4 Amazon.edu

For HW4, we'll use servlets to build a little bookstore called Amazon.edu. The conceptual structure of Amazon.edu is similar to the CGI homework, but usings servlets makes things a little easier. Amazon.edu is due midnight ending Thu June 1st. You may use at most four late days on this homework.

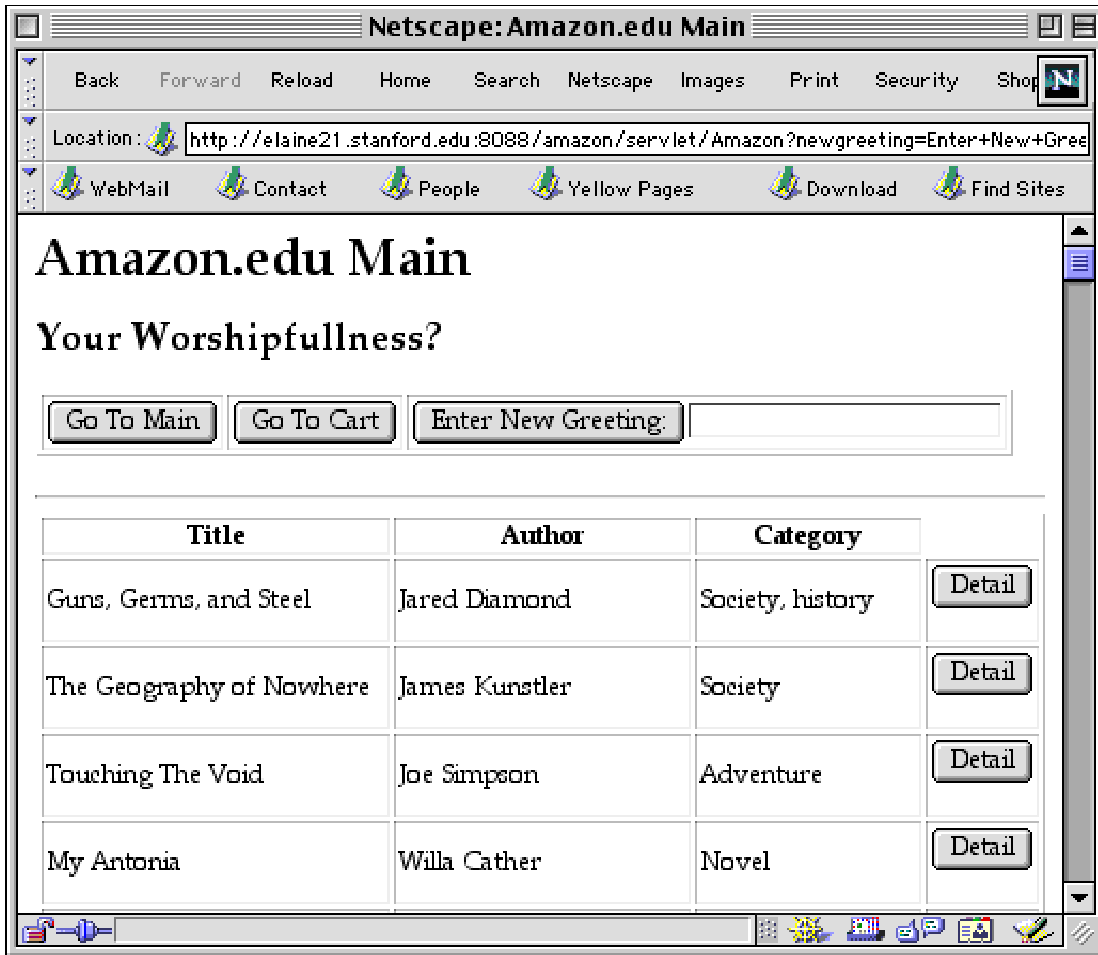
The first part of this handout describes how Amazon.edu works from the user's point of view. The second part gives some suggestions about how to implement it. The last section describes the logistics of running servlets and JSPs on leland. There are also some starter files on leland with code to get you started.

The solution will use CGI-type logic to create the main page and the cart page, and a JSP to create the detail page. (P/NC students may work in teams of two and use CGI logic to create the detail page instead of using JSP.)

1. Main

Amazon.edu has a simple 3-page interface. The first page you see is the "main" page which features

- The title and h1 "Amazon.edu Main"
- A greeting — in his case "Your Worshipfullness?" (the greeting is described below)
- "Go To Main" and "Go To Cart" navigation buttons (described below)
- An "Enter New Greeting" control that allows the user to type in the greeting they would like Amazon to use.
- A table listing all the books in the database by Title, Author, and Category (but omitting the description). There is a "Detail" button for each book. The top row of labels uses "<th>" while the others use "<td>".



The TextDB class provided for you takes care of the details of reading the book database into memory.

Persistent Greeting

Initially there is no greeting. The "Enter New Greeting" control sets the greeting and returns to the main page. That main page and all subsequent pages should use the new greeting. The greeting should be set in a persistent way so it works even when the user quits and restarts the browser. (Movie trivia: who is "your worshipfullness"?)

2. Detail

Clicking the detail button for a book leads to a page showing...

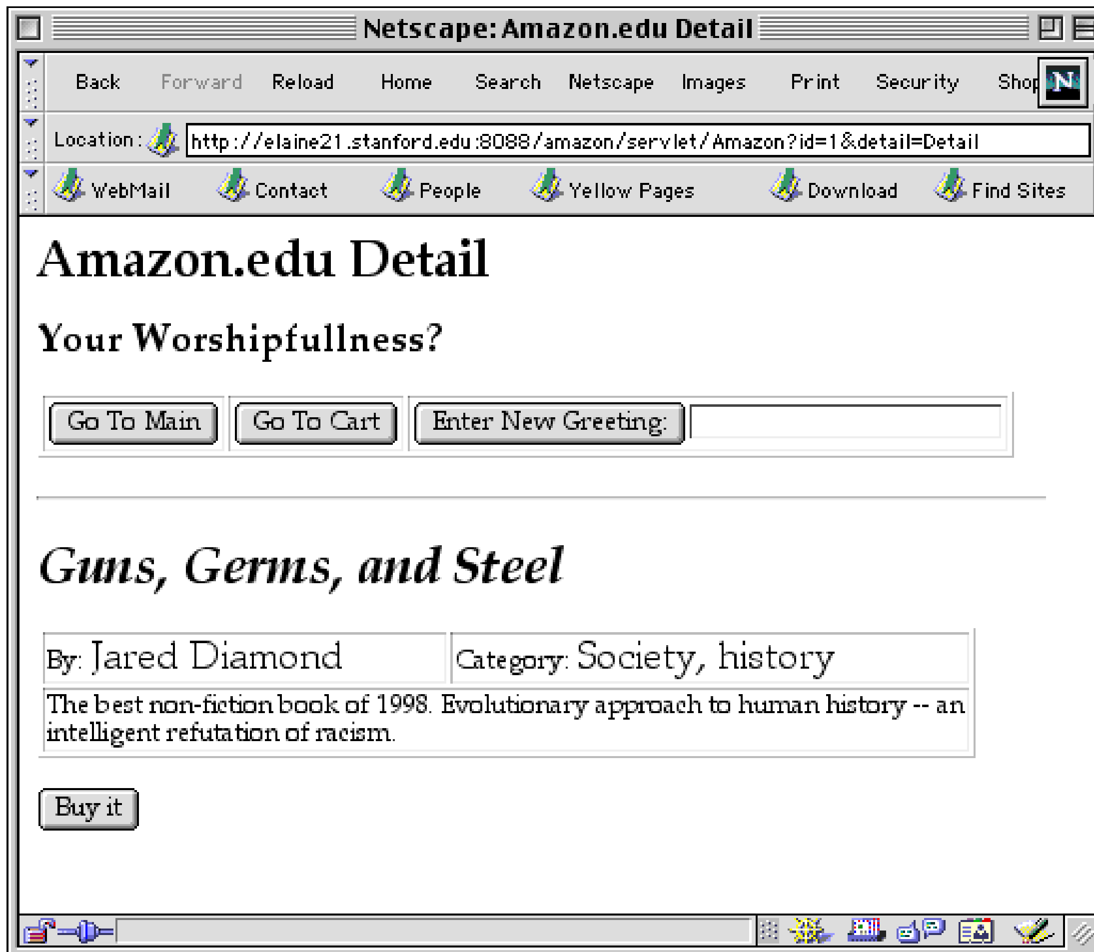
- The title and h1 "Amazon.edu Detail"
- The greeting and buttons again
- The information for that book presented with the following format...
  - The title is in a <cite> tag inside an <h2> tag

- The author and category are in two cells in the first row one of a table. Use ... to make the author and category fonts a little bigger.

- The description fills the second row of the table. Use <td colspan=2> to get the td to fill out both columns

• Finally, there's a "Buy it" button that can add the book to the cart

The detail page will be implemented as a little JSP. The JSP is well suited to building the nested HTML structure of the detail page.



Clicking the buy button adds the book to the session shopping cart, and goes to the shopping cart page (below) to show the cart state. Clicking the "Go To Cart" button from here and from the main page are both similar to clicking buy from here. The difference is that the buy button changes the cart state. Buying a book which has already been bought should go to the cart page but without changing the quantity— the only quantities allowed in our little binary shopping hell are 0 and 1.

3. Cart

The Cart page shows a table similar to the main page, but containing the subset of the books that have been bought, and with the word "bought" in bold in the rightmost column. The cart page shows...

- The title and h1 "Amazon.edu Cart" if coming from the Go To Cart button. If we are coming here because of a click on the buy button, the string "(Bought)" should be added to the titling to provide more rational feeling feedback for the buy button.
- The greeting and controls again.
- The table should show the current set of bought books — Title, Author, Category, and Description for each one. There should be an additional rightmost column that just says "Bought" for each book.
- The "Delete Cart" should change the cart to contain zero books and return to the main page.

Netscape: Amazon.edu Cart (Bought)

Back Forward Reload Home Search Netscape Images Print Security Shop

Location: http://elaine21.stanford.edu:8088/amazon/servlet/Amazon?id=1&buy=Buy+it

WebMail Contact People Yellow Pages Download Find Sites

Amazon.edu Cart (Bought)

Your Worshipfullness?

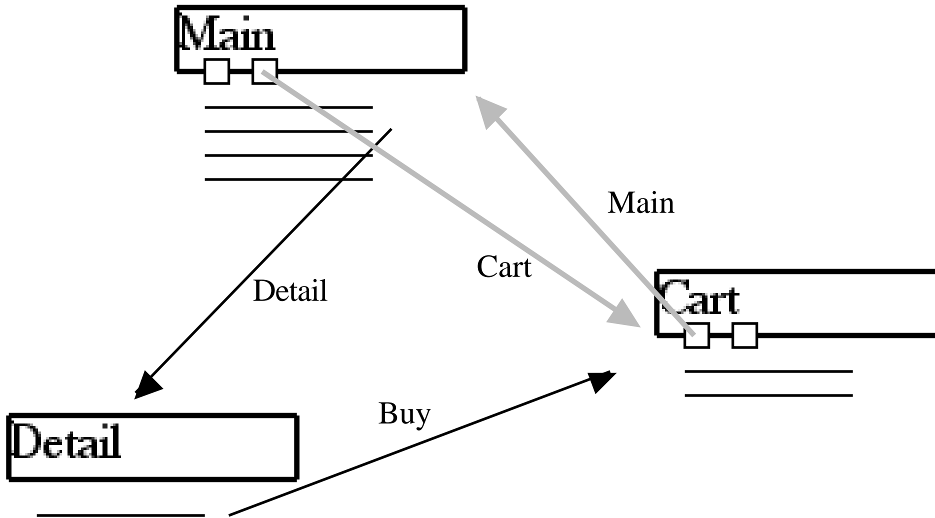
Go To Main | Go To Cart | Enter New Greeting:

Cart Contents...

Title	Author	Category	Description	
Guns, Germs, and Steel	Jared Diamond	Society, history	The best non-fiction book of 1998. Evolutionary approach to human history -- an intelligent refutation of racism.	Bought

Delete Cart

Navigation
Here's a map showing the most common transitions — the book buying path is in black, and the navigation button paths between main and the cart are in gray...



Implementation Ideas
As with the CGI homework, the solution will be to build a single servlet which looks at the request bindings to see what page to generate. Internally, the servlet session feature will be the perfect way to keep track of the current state of the cart.

doGet()
As with the CGI homework, the servlet needs to figure out which page it's supposed to send back. Use (`request.getParameter("buttonName") !=null)` to check for bindings in the request.

To keep things organized, separate out the doDetail(), doCart(), and doMain() methods to generate the individual pages.

System.out.println()
Use System.out.println("string") to print out debugging information. This will show up in your server terminal. It's fine to print out lots of stuff -- this is the only debugging tool we have.

Code Re-Use
There's great potential for of code re-use here. Don't be afraid to make up your own private utilities with a few parameters if the utility can then be called from several places.Create a utility method to generate the common HTML at the top of every page. Create a row-writing method that you can use to create the table rows for the main page and the cart page.

Session

The session is the perfect way to track the current collection of bought books. At the start of doGet(), detect if there is a session, and if not create a new empty one. The lecture example installed the "Tracker" object in the session. In this case, an even simpler strategy will work — just install a Vector object directly in the session since a single Vector is all that's needed to store the shopping cart state. In the Vector, store the row numbers of the bought books. Use session.setAttribute("string", object) to put things in the session.

Buying books and printing out the contents of the cart can use the cart Vector stored in the session. Every time the servlet is invoked, the same session will be present. As a result, the Amazon servlet should not store session specific state in traditional instance variables. The session object provides all the necessary storage for the servlet. This solves the traditional consistency problems with the forward and back buttons, and multiple windows aimed at the store at the same time (try it — they'll automatically share the session as with the lecture example).

Vector/String/int Operations

Since the database never changes, we do not need to do the nasty "id" scheme as in the CGI homework. The row number will be sufficient to identify each book. It turns out to be convenient to store and manipulate the row numbers in their String form, and convert them to int only when necessary. Here are some handy code snippets for Vector and String operations...

```
Vector cart = new Vector();

String num = "6";

cart.removeElement(num);    // remove an element equal to num if present
cart.addElement(num);       // add num to the end of the cart Vector

//// Do something with all the Strings in the cart Vector
for (int i=0; i<cart.size(); i++) {
   num = (String) cart.elementAt(i);
   // do something with num
}

cart.clear();               // remove all the elements from a vector




//// String/int conversions
int n = Integer.parseInt(num);    // figure the int value for a string
String s = Integer.toString(n);         // figure the string form of an int


//// String compare (both strings must be non-null)
string1.equal(string2)
```

```
//// Concat strings and ints using +
int n;
String string;

("hello" + string + n)      // Converts the int to string
                          // and concats it all together.
```

TextDB

The TextDB classes provided for you does the work of reading the text database into memory. The servlet should have a single "db" instance variable that points to a TextDB object that contains all the row data. The first time doGet() is called, create the TextDB object, passing the name of the file it should read from...

```
if (db == null) {
   db = new TextDB("books.txt");
}
```

The TextDB is created the first time and stored in the db ivar. For subsequent calls to doGet(), the db is ready. Your code can send two messages to the db to get data out of it...

- int size() -- the number of rows in the db.
- String[] getRow(int num) -- return an array of strings representing one row. Row 0 is the column headers for the database, and the subsequent rows are data.

Detail Page

There's a simple BookBean class provided for you. Create an instance of BookBean on the request for the JSP. Your JSP should be called "/detail.jsp". If there are errors in your JSP, the error messages generated will be quite cryptic. Compare your JSP with the lecture example very carefully to make sure your syntax is correct.

Greeting Cookie

Use a persistent cookie to remember the greeting. Check that it works even when you quit and restart the browser.

Setup Logistics

Here's how to create your tomcat/servlet setup. I apologize for yucky this setup is, but we will avoid the leland problems we had with HW3.

1. Java

Edit your .cshrc file in your home directory. Add the following line to set your java classpath.

```
setenv CLASSPATH /usr/class/cs193i/tomcat/lib/servlet.jar
```

Then, edit your unix path variable includes the directory "/usr/pubsw/apps/jdk-1.2.2/bin"

Here's what the relevant part of my .cshrc looks like afterwards...

```
# Now we actually set the path.
# We add your home directory and bin directory to the system list.
# You may add additional path customizations here.
set path=( \
    /usr/pubsw/apps/jdk-1.2.2/bin \
    /usr/class/cs108/staff/bin \
```

Do a "source .cshrc" to load the changes. Afterwards, you can check that it's done correctly...

```
elaine21:~/> which javac
/usr/pubsw/apps/jdk-1.2.2/bin/javac
elaine21:~/> echo $CLASSPATH
/usr/class/cs193i/tomcat/lib/servlet.jar
```

2. Directory Structure

In order to run servlets, your directories need to be set up just so — we'll use the /usr/class/cs193i/materials/servlet directory as a model. Use the following command from your home directory to copy the model servlet directory. The servlet directory can be located anywhere.

```
~/> cp -r /usr/class/cs193i/materials/servlet .
~/>
```

- The **servlet** directory contains the "books.txt" file. This is where you start and stop the server.
- The **servlet/conf** directory contains the server.xml configuration file
- The **servlet/webapps/amazon** directory contains an index.html that you may access at http://server/amazon/. This is also where JSPs go.
- The **servlet/webapps/amazon/WEB-INF/classes** directory contains all the .java and .class files.

1. Edit server.xml

Edit the server.xml file to use your own random port number. There are comments in the file showing you where to do this.

2. Server

CD to the servlet directory. Run the "startcat" command there. This should start the tomcat server and echo a lot of configuration. You will get an error message if an instance of tomcat is already blocked on your chosen port #. This terminal is

where you will see printlns from your servlet and other error messages. Use the stopserver command to stop the server.

3. Java Classes

Use a separate terminal down in /servlet/webapps/amazon/WEB-INF/classes to edit your .java files. Use "javac *.java" to compile them. If javac cannot be found, your unix path is wrong. If the servlet classes cannot be found, your java classpath is wrong (check the steps above).

4. Browser

From the browser, connect to your tomcat server...

http://server:port/amazon/ --- get index.html

http://server:port/amazon/servlet/Amazon --- run the servlet

The server should print a "Reload" message when it notices that you have re-compiled your java, and so it loads the new version.

5. JSP

When it's time for the detail JSP, put it in servlet/webapps/amazon alongside the index.html.

Why is this setup so complicated?

Tomcat is an industrial strength tool, and its design is not focused on exploratory little users like us. It is complicated to use because it is extremely configurable. It would be nice if there were a non-configurable "easy" mode built in where you could just run a couple servlets without all this structure. Remember how complex this setup was when you are developing your own tools. Remember: "Easy things should be easy. Difficult things should be possible." (In fairness, tomcat is open source, so it's possible for any of us to submit an "easy mode" back to the project.)

Deliverables

Put your readme in your servlet directory, and submit the whole thing.