CS108, Stanford Handout #33
Young

HW5 Web

In this assignment we'll build two separate web projects. This assignment is due the evening of Wednesday February 17th at 11:59pm. You cannot take late days on this assignment. I want all members of each project team ready to work on the class project which will be handed out the next day, not playing catch up finishing HW5 with late days. If you are not done at midnight, turn in what you have. We will have a draconian **per-minute** late policy that will rapidly turn your grade to **zero in just three hours** if you submit late.

In contrast to previous assignments, which could all be done using a single Eclipse "Java Project" for this assignment you'll want to create two separate Eclipse projects one for each part of the assignment. I'll walk you through the first project. Pay careful attention, because the second project you'll have to figure out a lot on your own.

For this assignment (and for the final project) you'll need to get a Enterprise Edition of Eclipse and install Apache Tomcat. See the "Setting Up Tomcat" handout for details.

Reminder: when creating web components use the specialized Eclipse creation tools, not the general ones. For example don't use "New > Java Project" use "New > Project …" and then choose "Web > Dynamic Web Project." Don't use "New > Class" to create a Servlet class instead use "New > Other …" and then choose "Web > Servlet."

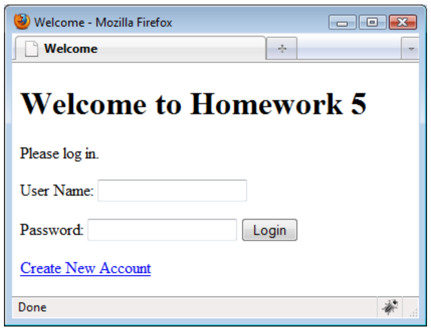
Use Eclipse to run the web server to test your project. You can right mouse-click on an HTML file, JSP, or Servlet that is part of your Eclipse project and then choose "Run As > Run on Server". This will start up Eclipse's internal web browser to display the webpage. Once the server is running, you can copy and paste the URL from Eclipse's internal test web browser to another web browser, such as Firefox, Chrome, or IE, to view your webpages using a different browser.

For the login system part of the assignment, place classes into the package login. For the store part of the assignment, use the package store.

User Login System

For the first part of our assignment we'll create a simple set of webpages which will allow users to login to a website. Our website will consist of the following webpages:

- A homepage where the user can either login or move to the new account webpage.



Welcome to Homework 5

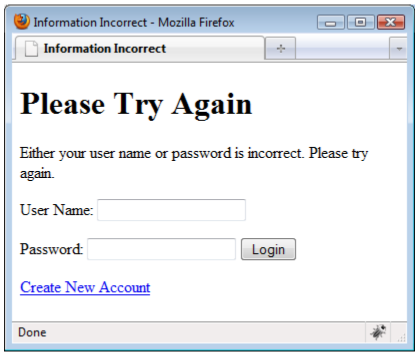
Please log in.

User Name:

Password:

Create New Account

- A webpage which shows up when the user has tried to login but entered an illegal name or password.



Please Try Again

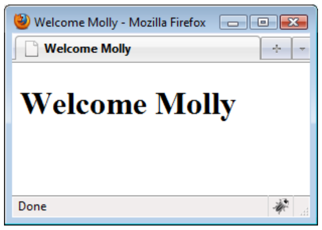
Either your user name or password is incorrect. Please try again.

User Name:

Password: Login

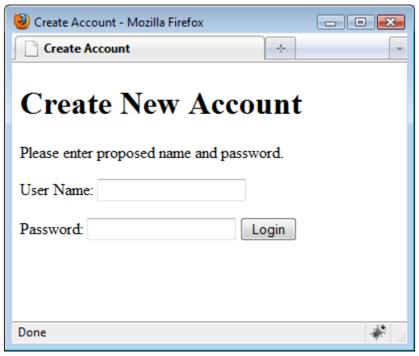
Create New Account

- A "user welcome" webpage where the user goes after a successful login. Note that the user's name is incorporated into the title and heading on this webpage.



Welcome Molly

- A "create account" webpage which allows the user to create a new account.



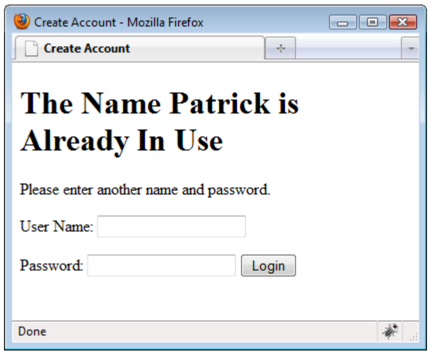
Create New Account

Please enter proposed name and password.

User Name:

Password: Login

- An "account name in use" webpage where the user is taken if they try to create an account and the account name they've requested is already in usage. If they try to create an account and the name is available, instead take them to the "user welcome" webpage above.



The Name Patrick is Already In Use

Please enter another name and password.

User Name:

Password: Login

Your system will include the following parts:

- Two servlets, one to handle logins, one to handle account creation.
- An account manager class, which keeps track of account names and passwords and allows the creation of new accounts.
- A class which is responsible for initializing everything in the WebApp.
- A bunch of HTML and JSP files which actually present information to the user.

Go ahead and create your project. Again a reminder, make sure you create a "Dynamic Web Project" not a standard "Java Project" and make sure that you are running the Enterprise Edition of Eclipse.

Account Manager

Create your account manager class. This is just a standard Java class. But don't put it in the default package, create a new package for it. In general, you don't want to put anything in the default package for your web applications. Anything in the default package cannot be accessed from JSPs, since there is no way to access something from the default package from outside of that package, and JSPs will not be created in the default package.

As previously noted, the account manager should keep track of names and corresponding passwords. In an actual system, the account manager would connect to a database. We'll keep things simple for this assignment, and instead simply store the information in the account manager object itself using a collection.

Write methods that allow us to check if an account exists, to see if a string matches the password for a particular account, and to create a new account. To support testing, your account manager should start with two accounts:

"Patrick" with the password "1234"

"Molly" with the password "FloPup"

We'll assume these are there when we test your assignment.

To keep things simple, you can go ahead and store the password values in "clear text." However, do be aware that this is not a good idea in an actual system. In an actual system, you should store encrypted passwords, using a technique similar to the one we discussed in the Hash Cracker part of the previous assignment.

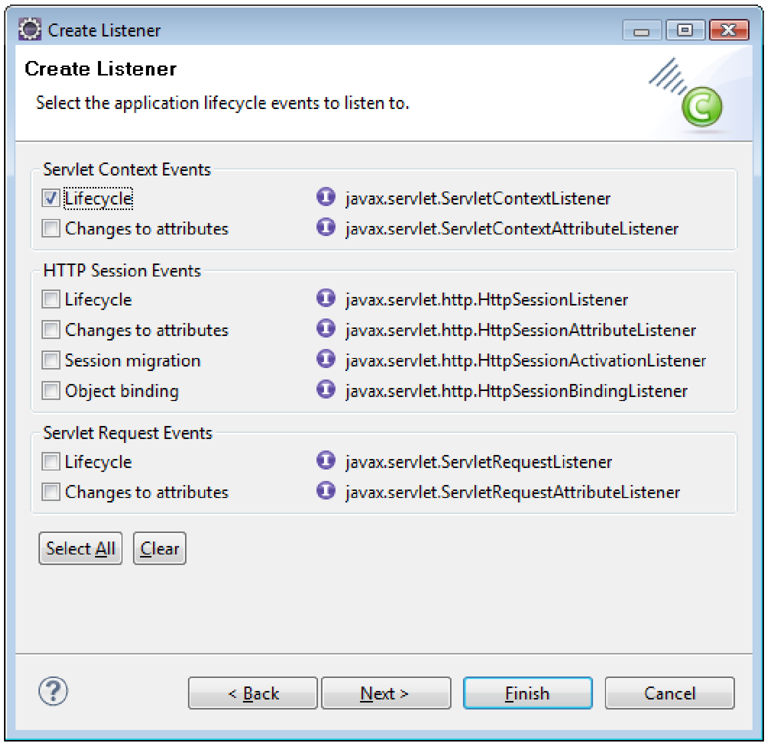
I recommend you stop at this point, and test your account manager using tests written with JUnit.

Initializing the Application

At this point, you may be tempted to write the login and account creation servlets. If you do, you'll discover a bit of a problem. The login and account creation both need access to the account manager. Who sets up the account manager and where is it stored?

As we discussed in class, there are several places where information can be stored. These include the ServletContext for information shared between all servlets and JSPs, the ServletConfig for getting initialization information to a specific servlet, and the session for sharing information within a single user session. In this case, we want to share the same account manager across all our servlets and JSPs so the ServletContext is the correct place to store our account manager.

We need to somehow create our account manager and store it in the ServletContext where everyone can access it. The correct way to do this is by creating a listener class. Go ahead and create a listener class in Eclipse. You can do this by choosing "New > Other …" and then going down to the list of web items and choosing "Web > Listener". Go ahead and give it a package and a name and hit the "Next>" button. You will then be presented with a list of items that your class can listen for:


Create Listener

Create Listener

Select the application lifecycle events to listen to.

Servlet Context Events

[✓] Lifecycle	javax.servlet.ServletContextListener
[ ] Changes to attributes	javax.servlet.ServletContextAttributeListener

HTTP Session Events

[ ] Lifecycle	javax.servlet.http.HttpSessionListener
[ ] Changes to attributes	javax.servlet.http.HttpSessionAttributeListener
[ ] Session migration	javax.servlet.http.HttpSessionActivationListener
[ ] Object binding	javax.servlet.http.HttpSessionBindingListener

Servlet Request Events

[ ] Lifecycle	javax.servlet.ServletRequestListener
[ ] Changes to attributes	javax.servlet.ServletRequestAttributeListener

We're interested in the Lifecycle for Servlet Context, so just check the top box and then click "Finish".

In the contextInitialized method from your new class, go ahead and create an instance of your account manager class. Get a hold of the ServletContext using getServletContext on the ServletContextEvent you received as a parameter, and then set the account manager as an attribute of the ServletContext. Servlets and JSPs will be able to get a hold of the same ServletContext object and access the attribute you've set to get a hold of the account manager you create here.

You would use the contextDestroyed method to do any cleanup necessary when the system shuts down.

Logging In

We’re now ready to add code for the login process. Start by creating the HTML file for the homepage. As seen on the first page of this assignment, this consists of an <h1> heading, some text, a form, and a link to the Create Account webpage, which we’ll create later.

Set the form up, like this:

```
<form action="LoginServlet" method="post">
```

So that when the form is submitted it calls the LoginServlet.

Now let’s go create the LoginServlet itself. Use “New > Other …” and then choose “Web > Servlet”. As your form uses the POST method, we need to override the doPost method in the servlet and can ignore the doGet.

In the doPost method, get a hold of the account manager which we previously created in the last section. You can do this by calling getServletContext and then retrieving it using the attribute you used in the last section.

Now get the name and password entered by the user using getParameter on the HttpServletRequest passed to doPost. Check with the account manager and see if they are correct. If they are, we’ll want to switch to the “user welcome” page, otherwise we’ll switch to the “Please Try Again” page. You can generate these pages dynamically using response.getWriter and then printing out the new webpage. However, you may find it easier to create them as separate HTML or JSP files and then transfer from the servlet to these files.

To do this, get a hold of a RequestDispatcher using request.getRequestDispatcher and then call forward on the RequestDispatcher. This will cause control to pass from the current Servlet and move to the HTML or JSP page specified in your getRequestDispatcher call.

Creating Accounts

Creating accounts works very similar to logging in. Create a servlet to handle creating accounts. Get a hold of the account manager from the ServletContext. If the account name requested is in use, move to the “Name in Use” webpage, otherwise move to the “Welcome” webpage.

Basic Store Website

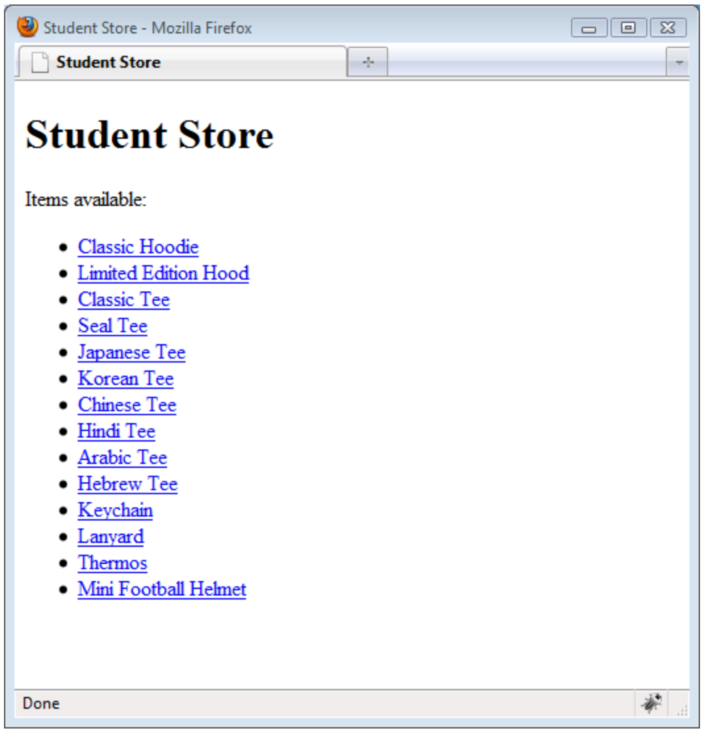
In this part of our assignment we create a basic store website. The webpage will work in conjunction with a MySQL database which will store product information. **To get Tomcat and MySQL working together, be sure to read the section marked "Getting Tomcat to Work with MySQL" starting at the bottom of page 9.** Use the provided products.sql file to manually setup your database using MySQL's command line interface (see the MySQL handout for details). Use the included MyDBInfo.java file (essentially the same as from Assignment #3, except placed in the "store" package instead of the "assign3" package) to store your account information. Once loaded, this is what the products table should look like:

```
+-----------+----------------------+--------------------+-------+
| productid | name                 | imagefile          | price |
+-----------+----------------------+--------------------+-------+
| HC        | Classic Hoodie       | Hoodie.jpg         | 40.00 |
| HLE       | Limited Edition Hood | LimitedEdHood.jpg  | 54.95 |
| TC        | Classic Tee          | TShirt.jpg         | 13.95 |
| TS        | Seal Tee             | SealTShirt.jpg     | 19.95 |
| TLJa      | Japanese Tee         | JapaneseTShirt.jpg | 17.95 |
| TLKo      | Korean Tee           | KoreanTShirt.jpg   | 17.95 |
| TLCh      | Chinese Tee          | ChineseTShirt.jpg  | 17.95 |
| TLHi      | Hindi Tee            | HindiTShirt.jpg    | 17.95 |
| TLAr      | Arabic Tee           | ArabicTShirt.jpg   | 17.95 |
| TLHe      | Hebrew Tee           | HebrewTShirt.jpg   | 17.95 |
| AKy       | Keychain             | Keychain.jpg       |  2.95 |
| ALn       | Lanyard              | Lanyard.jpg        |  5.95 |
| ATherm    | Thermos              | Thermos.jpg        | 19.95 |
| AMinHm    | Mini Football Helmet | MiniHelmet.jpg     | 29.95 |
+-----------+----------------------+--------------------+-------+
```

Note: As with assignment 3's metropolises.sql file, before loading the products.sql file make sure you replace the name of my database in the USE statement on the first line. You should use your own assigned database not my database, otherwise you will get an access error.

This website will consist of the following sets of pages:

- A homepage which lists all the products found in a MySQL database. Here is a screenshot of the homepage:[1]



Student Store

Items available:

- Classic Hoodie
- Limited Edition Hood
- Classic Tee
- Seal Tee
- Japanese Tee
- Korean Tee
- Chinese Tee
- Hindi Tee
- Arabic Tee
- Hebrew Tee
- Keychain
- Lanyard
- Thermos
- Mini Football Helmet

Each product listed in the homepage should be linked to a corresponding product page. The URL for each product page will be the same, except for productid information encoded into the URL. For example, here is a relative URL for the Classic Hoodie webpage:

```
show-product.jsp?id=HC
```

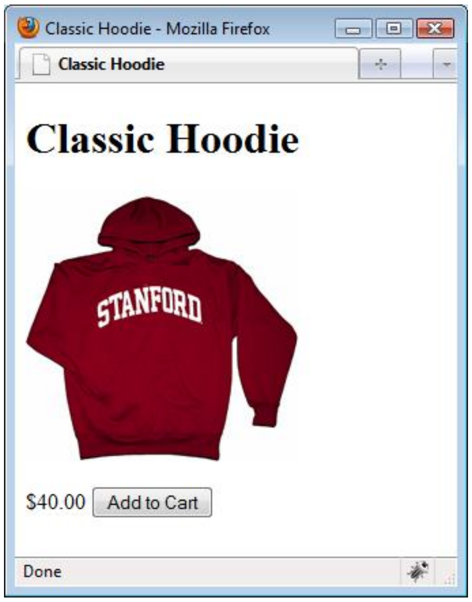
Whereas here is the URL for the Mini Football Helmet:

```
show-product.jsp?id=AMinHm
```

Product IDs used in the URLs match those in the SQL database.

- Product pages, where each product page is dynamically generated from information in the product database. The JSP generating the product page will strip out id encoded in the URL to lookup the product in the database. Here is the Classic Hoodie webpage:

[1] Create the student store list using the <ul> Unordered List and <li> List Item tags. An unordered list will show up as a bulleted list. Individual items in the unordered list are denoted using the <li> tag.

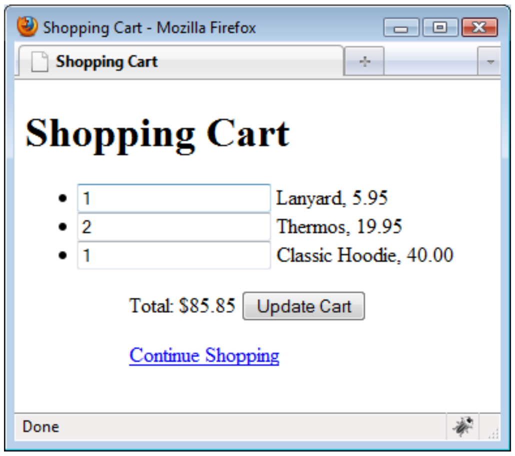


Classic Hoodie

$40.00 Add to Cart

Image files for each product webpage are provided with the assignment's downloads. Image file names are included in the database.

- When the user clicks on the "Add to Cart" button it takes them to the "Shopping Cart" webpage. Here's what the shopping cart looks like with a bunch of items in it:



Shopping Cart

- 1 Lanyard, 5.95
- 2 Thermos, 19.95
- 1 Classic Hoodie, 40.00

Total: $85.85 Update Cart

Continue Shopping

As you can see, the cart lists each item and includes a text field displaying the quantity of items being purchased along with the total cost of all items in the cart. If an item is already in the shopping cart, when the user clicks on the "Add to Cart" button, increment the quantity in the shopping cart. These quantity text fields are user editable, and if the user clicks on the "Update Cart" button, regenerate the shopping cart webpage showing the new quantities and the new total cost. Remove any items where the user has entered in zero as the quantity.

As you might guess, shopping carts are per user. That means if two people are visiting the website at the same time, they should each have their own shopping cart.

Things to Consider Before Getting Started

Think about the three webpages and how you want to implement them. In general, you have four basic choices:

HTML Only – Use this for static webages. As the homepage, product pages, and shopping cart page are all dynamically generated, this is not an appropriate option for any of the store webpages.

Servlet Only – Use this option when you have processor intensive code, and are generating simple HTML.

JSP Only – Use this option for complex HTML with only limited processing necessary in a few places on the webpage.

Servlet Combined with JSP – In this option, a servlet performs heavy processing and then passes intermediate results on to a JSP. This is a good option if you have processor intensive code and complex HTML.

Consider what helper classes you want to implement, if any, to support your Servlets and JSPs. You can find a list of the helper classes I implemented at the end of this handout.

Recommended Build Order

I recommend building your website as follows:

1. Start out by generating the homepage. This will allow you to experiment with getting the MySQL database to interact with the webserver. See below for notes on how to get the webserver to use the MySQL JDBC driver.

   Remember initialize data structures used across your website by creating a listener class as we did with the Login part of the assignment above.

2. Write the code which generates the individual product webpages. As this code is not product-specific, you'll need to be able to determine which product the user wants you to display. You can get this information out of the URL by using getParameter, the same way you can get information that the user has submitted from a form.

3. Implement your shopping cart.

Recommendations and Hints

Getting Tomcat to Connect to MySQL

In assignment 3, we connected a Java Swing program to a MySQL database. You may recall this required us to download a JDBC driver and add its JAR file to our project's build path. In this case, we need the web server, not our project, to use the JAR file. To do this, you'll need to find the location of where your Apache Tomcat installation keeps its JAR files. On my computer that is:

```
C:\Program Files\apache-tomcat-7.0.24-windows-x86\apache-tomcat-7.0.24\lib
```

Copy the JDBC driver downloaded for assignment 3 (see the handout on JDBC) into that directory.

Passing Product Information to the Shopping Cart

Each product page includes a form and submission button which starts up whatever code you've written to handle the shopping cart. How does that code know which product has been added to the shopping cart? One way you can pass the product information on to the shopping cart code is through the use of a hidden input field. For example, my Classic Hoodie webpage includes the following input tag:

```
<input name="productID" type="hidden" value="HC"/>
```

When the user submits the "Add to Cart" request, this hidden input information is passed along with it. I can check the request received for the value of "productID" and see that the product added has id of "HC".

Session for Shopping Cart

Shopping cart information should be stored as session information. Each user will have their own corresponding session, which means they'll have their own shopping cart. You can initialize the shopping cart using a session listener, the same way we previously initialized shared information using a listener on the ServletContext.

Testing Shopping Carts

The standard Eclipse web server does not allow access from outside of your own computer. Which leads to the question, how can we test shopping carts to ensure that different users have their own carts. The answer is to use different web browsers. If you access your website from Firefox and again from Chrome, these will be treated as different sessions, even if both browsers are running on the same computer.

Classes Used

In addition to Servlets and JSPs, my implementation included a Product class, a ProductCatalog class, a ShoppingCart class, a DBConnection class (which encapsulated creating a connection to the database, but returned java.sql.Statement objects directly to those wishing to communicate with the database), and classes to handle setting up the ServletContext and the Session.

Credits

Images for store problem are taken from The Stanford Store website: http://store.stanford.edu/. The Stanford Store is student-owned and operated and provides funds for ASSU. It can be found in Tresidder Union.