The final project will bring everything we have learned together, including unit testing, jquery, sql, escaping, formatting, etc…

For this project you will create an online bank that allows users to:

- View their transaction history.
- Withdraw money.
- Deposit money.

Of course, with it being an online bank, no real money could ever actually be withdraw or deposited.

There will be many parts of this project, each due at different times. See below for the full requirements and due dates.

Requirements

When the user navigates to your online bank, the landing page should welcome them and provide a means to login.
If the user does not yet have an account there must be a button/link that takes them to an account creation page.
An account, at minimum, must consist of:

- A first name
- A last name
- A user name, must be unique
- A password
- An email, must be unique

All five of the above fields are required, the user may not successfully create an user account without supplying a value for each field.
Validation is added, server side only, to ensure every field is filled out, and that the entered user name and email have not already been used. Display appropriate errors messages.

After the user account is created, the user is sent back to the welcome page so they can login.
Do not autolog the user in.

On login, the credentials, username and password, are validated to not be blank or whitespace only. The credentials are also checked against the database. Only a valid combination of username and password will allow a user into the system.

The users cannot get to any of the account summary, transaction history, withdraw or deposit pages without first logging in. If the user attempts to enter a URL that would normally take them to the summary page without first logging in they are redirected to the welcome page.

Once logged in the user is greeted, by name, and a summary of their past transactions is displayed in chronological order.

The transaction history will be a record of each withdraw or deposit and the corresponding amount. This transaction history will need to be stored to, and read from the database and consist of:

- The transaction date and time with hour/minute/second precision
- The transaction type (withdraw or deposit)
- The transaction amount
- The account total after the transaction.

When displaying the transaction history to the user, the transaction date is formatted as month/day/year hour:minute:second am/pm, any amounts are formatted as $1,234.56.

From the summary page the user can choose to withdraw or deposit funds.
The mechanism used to allow a user to enter the amount to withdraw or deposit must be submitted to the server using AJAX.

The amount the user entered is validated as follows:

- Is numeric.
- Is greater than zero. (a zero value is invalid)
- If withdrawing, it cannot be greater than the current account funds.
- If depositing, it cannot be greater than $10,000.00.
    - This is for a single deposit, the account itself can get over $10,000.00 if multiple deposits are done.

If the user enters a value that does not meet any of the above validation they are presenting with an appropriate error message. This error message must be specific to what they did wrong, a generic message of "Invalid Number" is not acceptable.
This validation must be done on the server, this is financial information so client side validation is not sufficient.

The "Account" object, aka the Model of MVC, does the above validation, minus the "Is numeric" check which should be in the servlet.
For example, the Account object could have withdraw and deposit methods, these methods would then do the appropriate validation.
If an invalid number is provided the appropriate exception is thrown.
The Account object's amount is never set to an invalid amount, such as a negative value.

I suggest using BigDecimal instead of double for any numbers that represent money. The double data type suffers from floating point calculation rounding errors, whereas BigDecimal does not.

JUnit tests are created to test the Account object's business logic, and will consist of:

- Testing Withdrawing funds

Testing Depositing funds
Valid numbers (> 0)
Invalid numbers (<= 0, > 10,000 for deposit)
Invalid numbers (> current funds available for withdrawing)

The unit tests should verify the account amount was modified correctly, or that the appropriate exception was thrown in the case of invalid numbers.

The Account information is written to, and read from the database and stores the following:
An account ID, must be unique, can be an Identity column (and PK).
The account owner, FK to the users table (to the user name).
The account balance.
The last modified date.

When a user creates a new user account with the bank, two entries to the database will be written, one for the user itself, and another for their account. The starting balance should be zero.

Once a successful withdraw or deposit is done the database is updated to record the current account balance and last modified date.
A record in the transaction history table is also added to track the activity (the withdraw or deposit).

The transaction history presented to the user is refreshed from the database to show the most recent data, including the withdraw/deposit that was just performed. This is done using AJAX, this may be the same AJAX call as the withdraw or deposit or a new AJAX call, it is up to you.
The same ordering and formatting rules from above apply to the refreshed data.

EL statements should have the appropriate escaping.
This doesn't mean all EL statements.

All currency amounts displayed to the user are formatted as $1,234.56.

All database queries are protected from SQL Injection attacks.

All pages are protected from XSS attacks (injecting HTML or Javascript that gets executed).

The UTF-8 Character set is used for page encoding in JSPs. This means the user could create an account with an accented character, such as "é" if they desired.

The web pages should have some styling applied to them, plain white pages with only black text won't cut it this time.
The styling can be done using custom css and images, a JQuery theme or a combination of JQuery and custom css/images.
The styling won't be a big part of the grade, and I am not looking for anything fancy, just an attempt at making it look good.

Due Dates

Various parts of this project will be due at different times.
The same rules for late assignments will apply to each of the due dates below, points for the items below will be reduced if handed in late.
Each of the items below will have a separate dropbox on Angel in the Final Project folder.
The 100% completed project is due by 04/30/2014 by the end of the class period (9:45pm).
This includes both demonstrating the project and submitting the code.

Mockup	Due: 04/09/2014
A basic visual mockup of how you plan to layout your web pages is created. This can be done in any tool of your choice, or even something simple such as Microsoft Paint. The mockup should just give me, and you a basic idea of how you envision designing your web pages. The welcome page, and summary pages must be accounted for. Think about where you want the login area, how your welcome screen should look, how the summary page may look, where the withdraw/deposit controls wil go, etc… Keep in mind this is a mockup, if your final design changes that is fine.	

Account object and JUnits	Due: 04/09/2014
The "Account" model object is created. One or more JUnit test files are created that test the Account object's withdraw and deposit functionality. Valid and Invalid numbers are tested, the JUnit tests expect the appropriate response or exception. See the requirements above for more information. If your Account object needs to be changed later during the project that is fine. If that does happen, the JUnits should be adjusted along with it.	

Database Design	Due: 04/16/2014
The table structures to store the user information, account information and transaction history are provided. This can be a SQL file with the create table statements, or an Entity Relation Diagram (ERD). For the final project submit, the SQL statements used must be submitted along with the project code. If your database table structure needs to change later during the project, that is fine.	

Rough Draft	Due: 04/16/2014
Your NetBeans project code is zipped up and submitted to the rough draft dropbox. This will be a rough draft, I am not expecting it to be fully finished. This is roughly the halfway point, so you should be about 50% done. I would like to see the progress, and have a chance to give feedback on where you are to give tips or suggestions. Of course, if you have any questions, feel free to ask me during lab time. Don't wait until the rough draft is due if you have questions. As long as you have a good start to the project you can think of this as free points and feedback.	

Final Project	Due: 04/30/2014
All of the above requirements are met. The project is demonstrated to me. The project code is zipped up and submitted to the Final Code dropbox. ● NetBeans project code ● SQL statements for the database structure	

Extra Credit (10 points)
Allow users to create multiple banking accounts, so that a single user account can have funds in multiple bank accounts.
When a user logs into the bank application instead of seeing the transaction history they see a listing of their bank accounts and how much money is currently in each.
Clicking on a bank account will present the user with the transaction history for that account.
This also means, in addition to being able to withdraw and deposit funds into an account a user can transfer funds from one account to another.
The transfer feature should only be available if the user has two or more accounts. When transferring funds a user must select a source account, where the funds come from, and a destination account, where the funds are going.
The same limitation of $10,000 applies to transferring, a user cannot transfer more than 10,000 at once. An amount greater than the source account's total cannot be used.
WIthdraw, deposit and transfer are still submitted using AJAX.
The extra credit can be demonstrated and submitted along with the Final Project (04/30/2014)