Enterprise Programming - Evaluation

Code Quality

This section will have no set format, but is a collection of examples of sound software engineering techniques.

Refactoring Methods

When carrying out AJAX requests, I found that I was typing the same ajax request over and over, with slight variations between addresses, formats and function calls. I written a function that can be called to complete this task in a more elegant way.

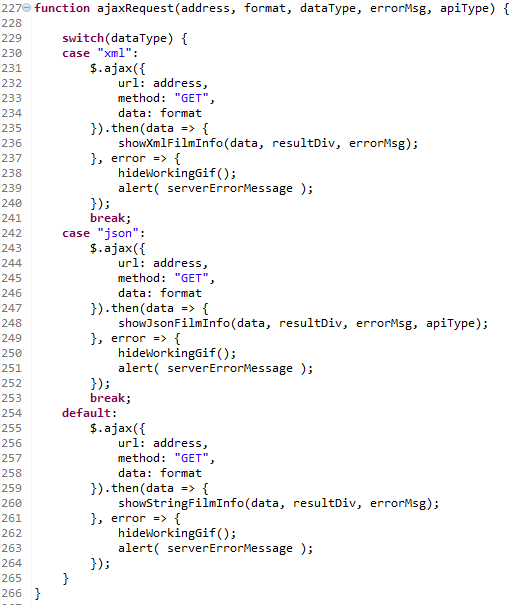


Figure 1 - AJAX function FilmSoap/WebContent/scripts/myjquery.js line 227

The function is called in every point in the code an AJAX request is required. The function takes in address, format, dataType as arguments to the AJAX call and errorMsg, apiType to deal with error checking and messages. A switch statement is used to select the appropriate function to deal with the data returned by the request (type of data returned impacts how the table is built). This is maintainable code as in the future another datatype could easily be added as another case in the switch statement.

The function reduces the amount of code needed in the individual methods as shown below.

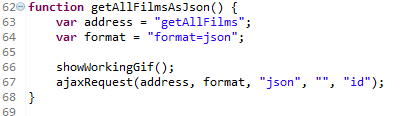


Figure 2 - Example of reduction in lines FilmSoap/WebContent/scripts/myjquery.js line 62

This reduces the amount of lines written and improves maintainability. For example, if another success handler is needed then this can be added into ajaxRequest() rather than updating every bit of the code that uses that request.

Variable Names and Declarations

Within my JavaScript code I found that I was typing the same error messages over and over. I chose to declare some error messages at the top of the file to be used throughout, as shown below.

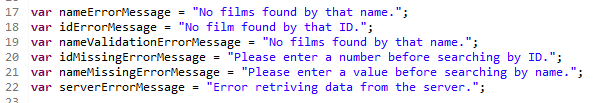


Figure 3 - Error messages FilmSoap/WebContent/scripts/myjquery.js line 17

These were all named as clearly as possible so that the content of the string is obvious when reading a method that uses this. The following naming convention was used

<type of error>ErrorMessage = “message”;

The ErrorMessage ending may seem redundant but improves readability when looking at a function that uses it. This is shown in figure 4.

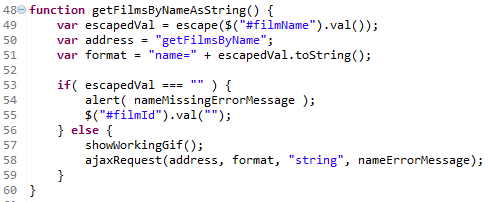


Figure 4 - Error Message variable used instead. FilmSoap/WebContent/scripts/myjquery.js line 48

As you can see the ajaxRequest function is much cleaner to read and if the error message content needs to change, this is done at the declaration at the top of the file rather than every instance. It also keeps the line short as lengthy error messages can cause it to spill onto two lines.

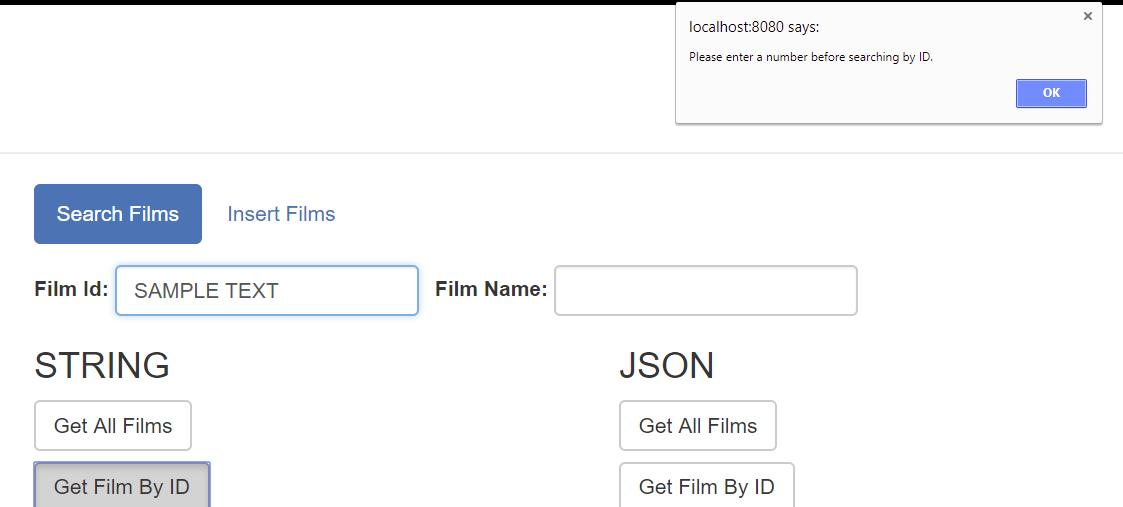
Validation

When the user is required to input information into the form, some client side validation can be used to improve the user experience and can ensure the correct types of data are being sent to the server.

I’ve used validation to do the following:

* Ensure user has entered a value when trying to search by id/name
* Ensure the correct type of data is used for both
* Ensure user has entered a value when trying to insert film
* Ensure the correct type of data is used for the relevant field

An example of this is shown in figure 5.



This is achieved by performing a regex check on the value entered. This is shown in figure 6.

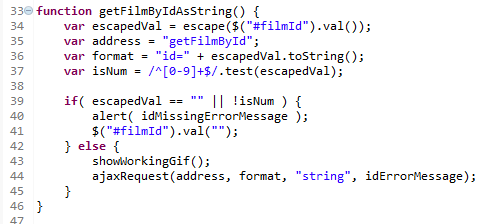


Figure 5 – Regex Check for number. FilmSoap/WebContent/scripts/myjquery.js line 33

This is shown to be especially useful when enforcing the correct input when trying to insert a film. When the insertFilm() function is called, checkFormIsValid() is called to ensure the correct input. checkFormIsValid() is shown in figure 6.

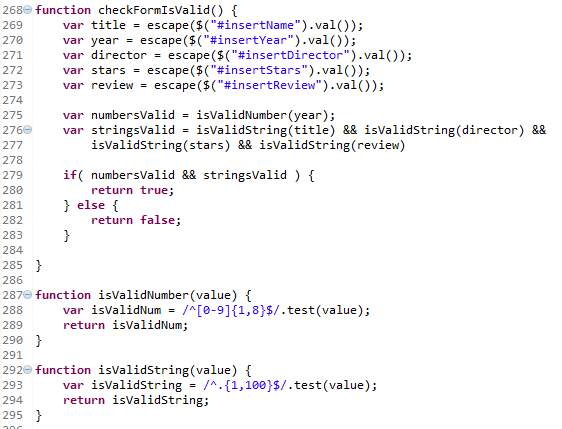


Figure 6 - Check form is valid FilmSoap/WebContent/scripts/myjquery.js line 268

The length values that the regex enforces match the rules defined in the MySQL table. However, someone who is malicious may be able to send a call directly bypassing the browser validation. The real issue would be trying to enter a string in the year field, so I added a try/catch to stop the attempt if the year is unable to be parsed into an int. This is shown in figure 7.

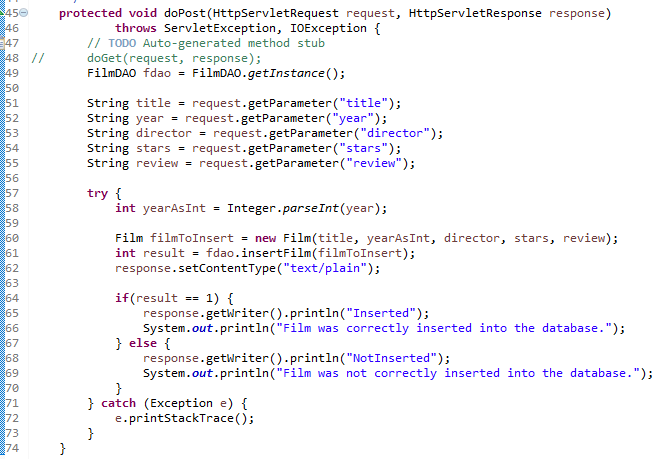


Figure 7 - Server side validation - FilmSoap\src\coreservlets\InsertFilm.java line 45

Admittedly, this probably isn’t as robust as what is required in a production setting, but stops and returns an error before the attempt to insert a string into a int column.

Connection closing

One thing I noticed from the base FilmDAO class provided was that closing connections was performed within the try section of a try-catch statement. This is a problem because if an error occurs, the connection has been opened but does not get closed again. I moved the closeConnection() method into a finally block of the try-catch-finally, as shown in figure 8.

Not closing connections can lead to the MySQL server reaching its connection limit. This is because although the connection closes after each script, the connection stays open for the duration of the processing. If there is a lot of processing going on with the data then the connection may be open for a while. This means the amount of connections will grow as requests take longer, possibly leading to maxing it out.

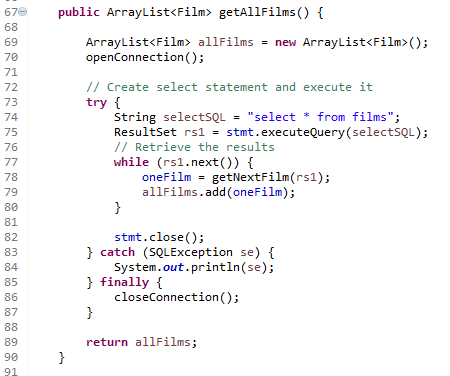


Figure 8

Use of libraries

I made use of the following libraries to save time and trouble ‘reinventing the wheel’.

To aid with XML generation, JAXB was used (figure 9):

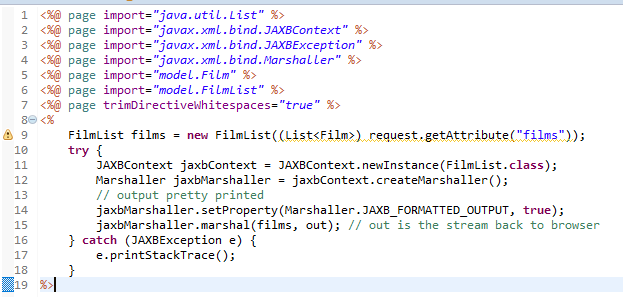


Figure 9

This just takes a filmList object and creates xml based on the annotation provided in the FilmList class, as shown in figure 10.

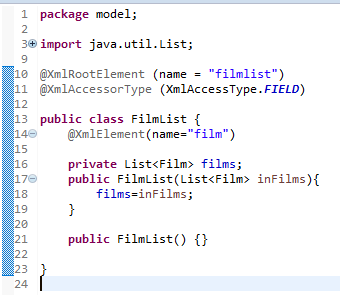


Figure 10 - FilmSoap\WebContent\WEB-INF\results\films-xml.jsp

Another example of this would be the use of Gson to aid with the creation of Java Objects into JSON, as shown in figure 11.

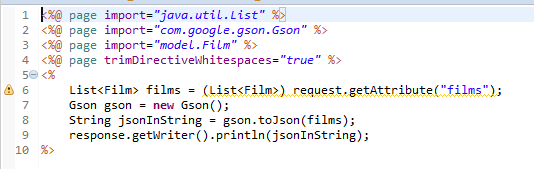


Figure 11 - FilmSoap\WebContent\WEB-INF\results\films-json.jsp

All this piece of code does is takes in a list of films are returns the JSON equivalent as a string.

Design Patterns

MVC

The model-view-controller design pattern is one in which different classes take over operations related to the application domain (the model), the display of the application’s state (the view), and the user interaction with the model and the view (the controller).

I’ve implemented this through my project layout and design. The ‘model’ in this case would be my Film.java, intended to model the entity stored in the relational database. The view would be the JSP’s which process the raw data provided by the servlets. This is able for viewing in the client app created in Web-Content. The controller would be servlets which take then requests and deal with them appropriately.

Singleton Pattern

When using the FilmDAO, I realised this would be a good place to implement a singleton design pattern. A singleton pattern restricts the instantiation of a class to only one object. This is useful because without it multiple FilmDAO objects could exist, all trying to send the same request. I refactored FilmDAO.java to the following (figure 12).

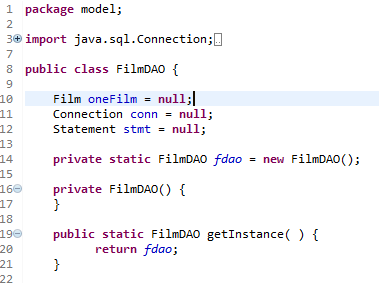


Figure 12 – FilmDAO.java

Simply, rather than creating a new object each time the filmdao is needed, FilmDAO.getInstance(); is called instead which returns the single existing instance of fdao. Fdao can then be used as normal to access the database.

Git/Version Control

Throughout the development of the project, I used my personal GitHub repository to host my project. This helped me greatly for a few reasons. Firstly, I switch between working on a desktop PC at home and a laptop out of the house. If I complete work on one machine, at the end of the working session I just commit and push my work to the master branch. This allows me to switch machine and instantly get the work I’ve completed without trouble.

Another benefit is that I get to revert to previous commits if something goes wrong. In addition, the git log helps to show how development taken place, as shown in figure 13.

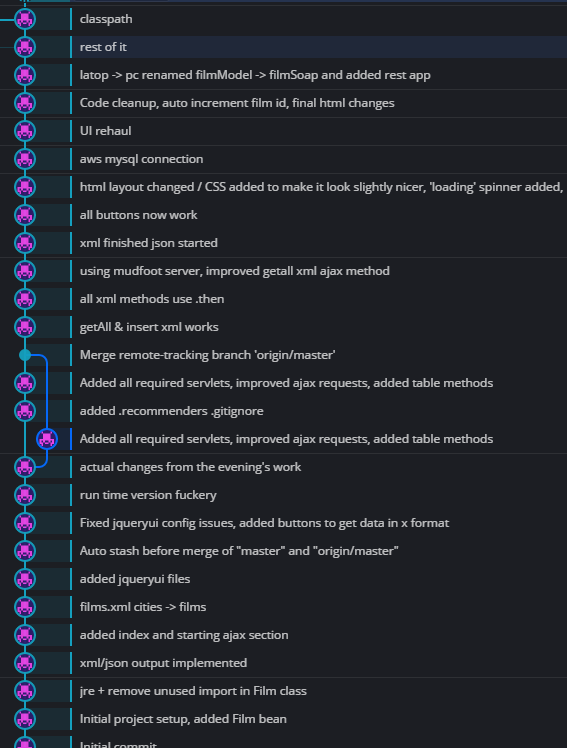


Figure 13

Bootstrap

To aid with user interface design, I used bootstrap classes. This can be seen within the code itself or by using the application. I used this because I personally am no good at making things look nice, when bootstrap have already done that for me. Figure 14 shows this.

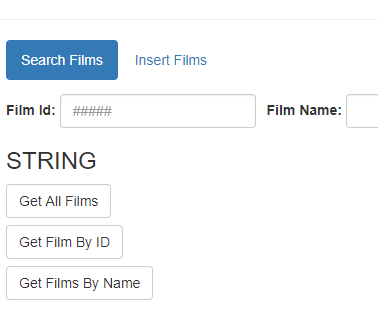


Figure 14

You’re even able to switch tabs to insert a film, reducing the amount of screen space wasted on forms (figure 14).

