COMS 417: Software Testing (Fall 2017)

LAB 5

Web Server Testing with HttpClient (Fluent)

Due December 6, 11:59 pm

Objective and Background

In this lab you will practice writing junit tests that exercise and test a “remote” server. (In fact, the server will be running locally on your development machine.) This lab will give you a chance to gain some concrete experience with:

- basic aspects of the HTTP protocol (methods, parameters, headers, status codes, and body content.)
- deploying and monitoring a modern java servlet engine.

You should come face to face with these typical web-testing challenges:

- Each test cycle can be long if you need to restart and redeploy frequently.
- Errors can be subtle and may sometimes be exposed only by certain sequences of requests.
- Each test may require you to fetch a sequence of pages in order to get to the state that will generate the behavior you are trying to test. Thus test execution can be noticeably slower than in the local application domain.

The web application you are testing implements five different dynamic endpoints. We have also supplied a static HTML page to facility testing one of the endpoints. You can think of this web application as being a prototype of a web service that maintains information about individuals of various species who reside on various planets. It is only a prototype because it can only remember one “record” at a time. Still, it shows most of the behavior that someone using such a database might see.

The primary endpoints are:

endpoint	**behavior**
command/clear.jsp	clears/resets the “database”
command/update.jsp	creates and populates a record describing a single individual
query.jsp	retrieves a “token” that authorizes you to retrieve the current contents for a particular planet
resultset.jsp	retrieves the resultset associated with a token retrieved from query.jsp

The web server also hosts an html form “postupdate.jsp” that lets you conveniently post a new record to update.jsp. This form is only for manual testing (i.e., to help you figure out how things work.)

Normal usage scenario:

```
 fetch Get clear.jsp to clear state.

 one or more times

     update variables (post to update.jsp)

     zero or more times

             fetch a token for a particular planet. (get to query.jsp)

             use the token to retrieve the result set for that planet. (get to resultset.jsp)

     end loop

end outer loop
```

Hints:

Knowing what you are sending to the server and getting access to the details of what the server is returning will be a central challenge in this project. Part of your planning needs to be to identify a way to get all the details you sent the server. Some echo service can be very helpful instrumentation for this need.

To get a good understanding of what is coming back, you may find it helpful to create some post and get helper functions (from HttpClient) that you can use to replace a normal test request so that you can “dump” all of the response components.

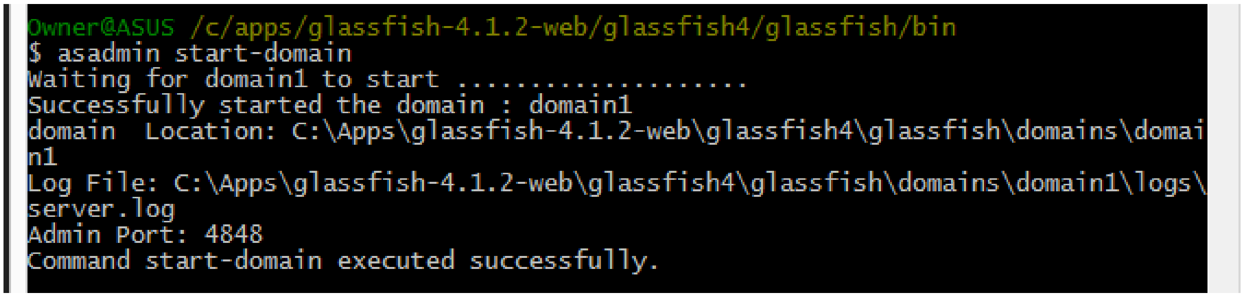
Wireshark and curl can help, but (especially wireshark) can be very timeconsuming – it’s a big learning curve investment for the few places in this project where it might actually help you.

In step 9 below, you retrieve a token from a log file. Don’t get confused by this manual kludge; it is *not* part of the normal use scenario. Your tests *will not* need to look at the remote log file; your tests will get the token directly from a header in the HTTP response. However, when using the browser to manually interact with the web server, you can’t see the header – so during manual testing you need extra instrumentation at the server. A richer client application (pages with javascript) running in the browser could use an AJAX style request to get access to the headers.

Lab Outline

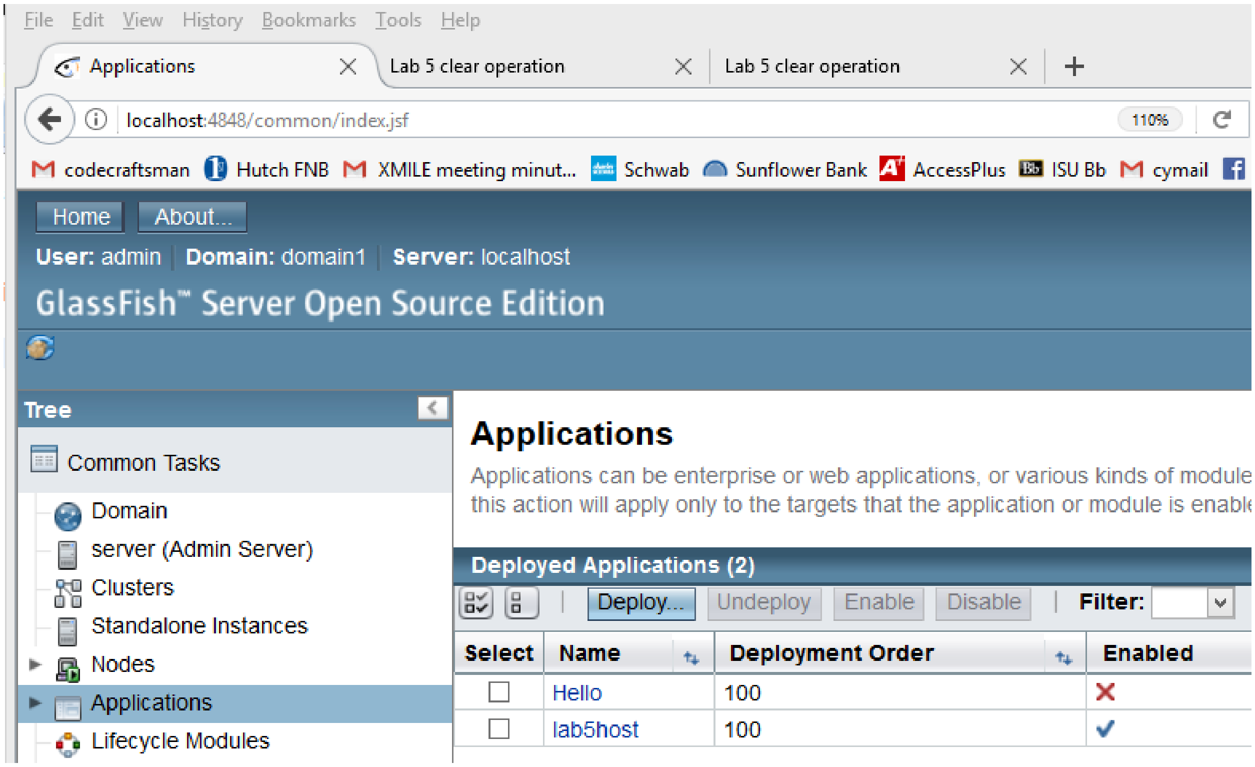
I. Get the webapp running.

1. install glassfish (or, at your own risk and cost, some other container of your preference.)
2. start glassfish from the command line. pay attention to the startup message; it tells you where to look for log files. The start up message should look something like this:



```
Owner@ASUS /c/apps/glassfish-4.1.2-web/glassfish4/glassfish/bin
$ asadmin start-domain
Waiting for domain1 to start ...................
Successfully started the domain : domain1
domain  Location: C:\Apps\glassfish-4.1.2-web\glassfish4\glassfish\domains\domai
n1
Log File: C:\Apps\glassfish-4.1.2-web\glassfish4\glassfish\domains\domain1\logs\
server.log
Admin Port: 4848
Command start-domain executed successfully.
```

   and reminds you of important information: what port you use to find the glassfish admin console, and where to look for logs.
3. Download cs417Lab5.war from the assignment page in Blackboard.
4. Rename the war file so that be deployed with the correct document root.
5. Using the admin console (as in the figure below), deploy the renamed war into glassfish at “localhost:8080/lab5host”.



User: admin | **Domain:** domain1 | **Server:** localhost

Applications

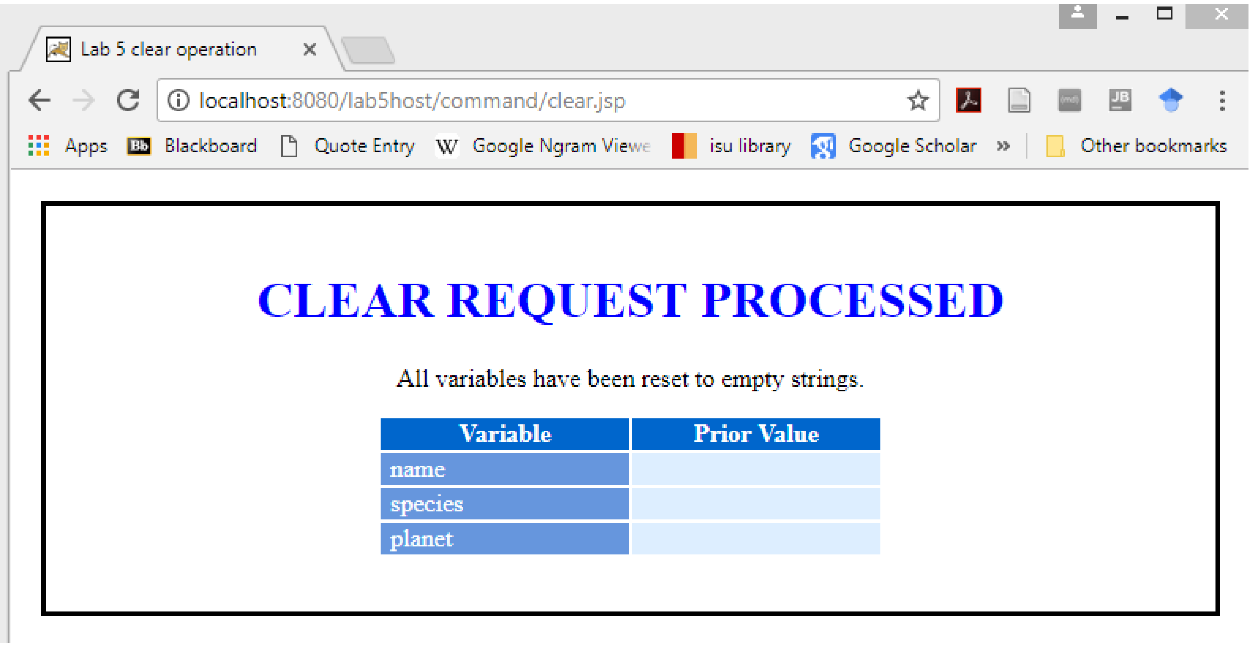
Applications can be enterprise or web applications, or various kinds of module
this action will apply only to the targets that the application or module is enable

Deployed Applications (2)

Select	Name	Deployment Order	Enabled
☐	Hello	100	✗
☐	lab5host	100	✓

6. Use a browser to walk through the usage scenario given above.

A request to http://localhost:8080/lab5host/command/clear.jsp should produce:

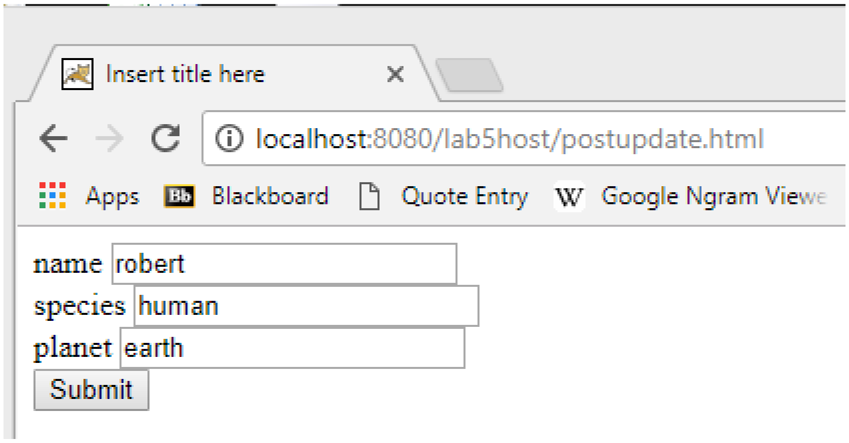


CLEAR REQUEST PROCESSED

All variables have been reset to empty strings.

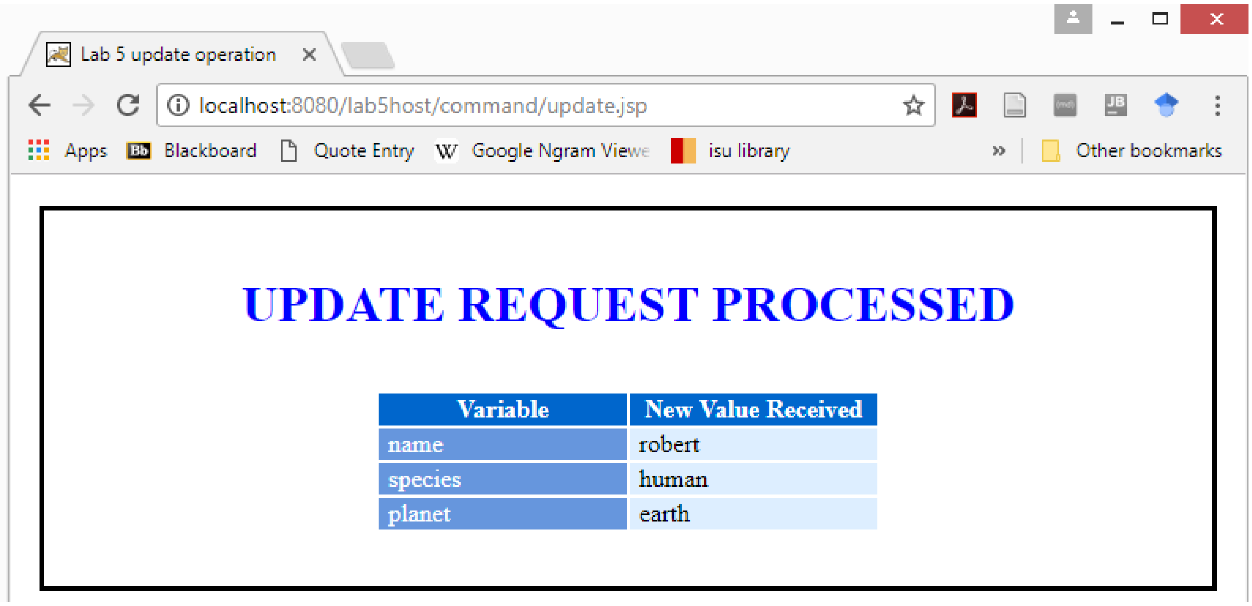
Variable	Prior Value
name	
species	
planet	

Following the clear with http://localhost:8080/lab5host/postupdate.html should get you the primitive form:



name robert
species human
planet earth

Submitting this form should retrieve the following page:

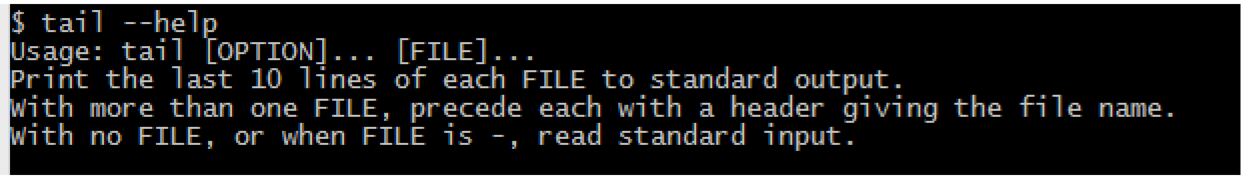


UPDATE REQUEST PROCESSED

Variable	New Value Received
name	robert
species	human
planet	earth

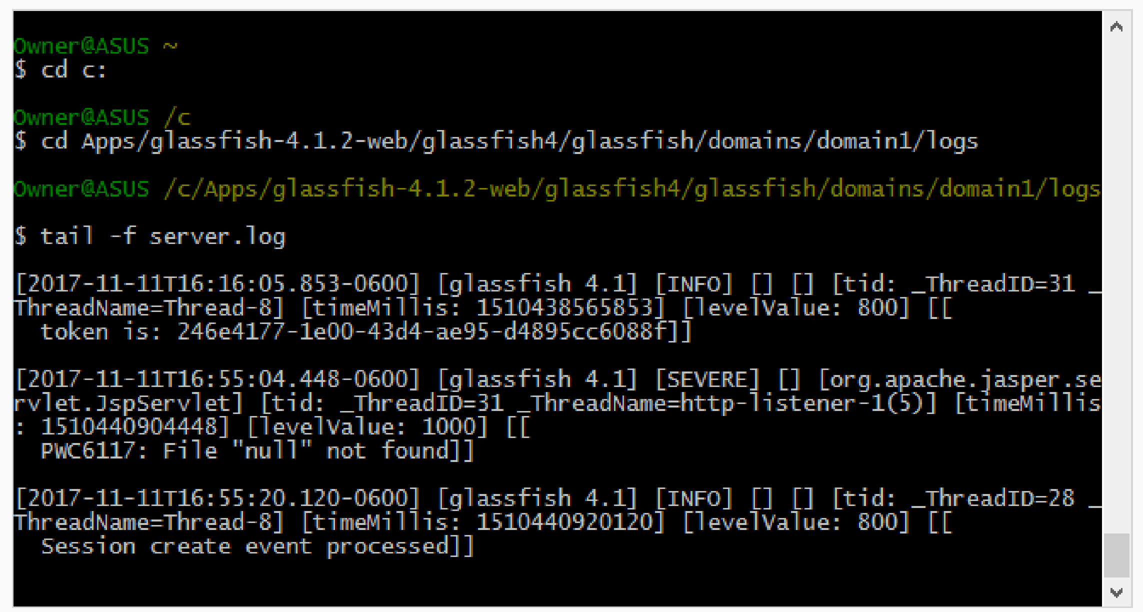
To perform the last two steps you need to be monitoring the server log file. This monitoring will be most convenient with some editor or viewer that can “follow” the log file, that is, that is capable to continually monitoring the file and displaying updates.

7. If you don’t have something equivalent to “tail” installed, find and install git-bash. You can tell whether you have tail by trying “tail –help” at the command line. You should get a response that begins with something similar to this result:



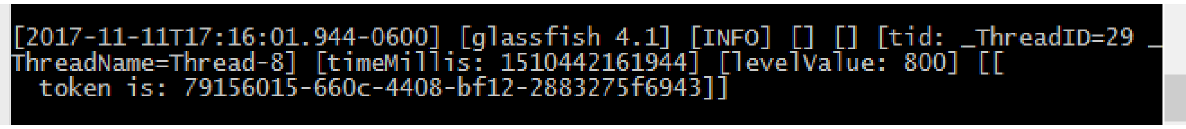
```
$ tail --help
Usage: tail [OPTION]... [FILE]...
Print the last 10 lines of each FILE to standard output.
With more than one FILE, precede each with a header giving the file name.
With no FILE, or when FILE is -, read standard input.
```

8. Once you have tail available, invoke “tail –f” to follow the log file location reported in step 2:



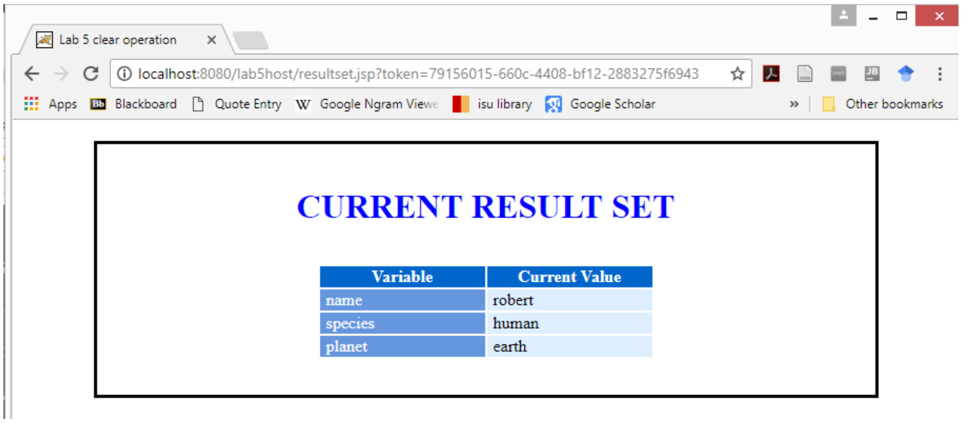
```
Owner@ASUS ~
$ cd c:

Owner@ASUS /c
$ cd Apps/glassfish-4.1.2-web/glassfish4/glassfish/domains/domain1/logs

Owner@ASUS /c/Apps/glassfish-4.1.2-web/glassfish4/glassfish/domains/domain1/logs

$ tail -f server.log

[2017-11-11T16:16:05.853-0600] [glassfish 4.1] [INFO] [] [] [tid: _ThreadID=31 _
ThreadName=Thread-8] [timeMillis: 1510438565853] [levelValue: 800] [[
  token is: 246e4177-1e00-43d4-ae95-d4895cc6088f]]

[2017-11-11T16:55:04.448-0600] [glassfish 4.1] [SEVERE] [] [org.apache.jasper.se
rvlet.JspServlet] [tid: _ThreadID=31 _ThreadName=http-listener-1(5)] [timeMillis
: 1510440904448] [levelValue: 1000] [[
  PWC6117: File "null" not found]]

[2017-11-11T16:55:20.120-0600] [glassfish 4.1] [INFO] [] [] [tid: _ThreadID=28 _
ThreadName=Thread-8] [timeMillis: 1510440920120] [levelValue: 800] [[
  Session create event processed]]
```

9. Now you have all the instrumentation you need to be able to perform a query and capture the token it returns. However, it might be a good idea, at this point, to repeat steps 5 and 6 now, so that you can see what the container is logging. To perform a query for planet earth records, request with http://localhost:8080/lab5host/query.jsp?planet=earth. You will get a blank page in the browser (the return body is empty), but the server should write the token it sent in the response header to the log file:



```
[2017-11-11T17:16:01.944-0600] [glassfish 4.1] [INFO] [] [] [tid: _ThreadID=29 _
ThreadName=Thread-8] [timeMillis: 1510442161944] [levelValue: 800] [[
  token is: 79156015-660c-4408-bf12-2883275f6943]]
```

10. Copy this token into the result set url to fetch the data in the browser: http://localhost:8080/lab5host/resultset.jsp?token=79156015-660c-4408-bf12-2883275f6943



CURRENT RESULT SET

Variable	Current Value
name	robert
species	human
planet	earth

II. Figure out what to test.

1. The appendix to this document contains the requirements/documentation for the web app. Remember, the web-app is the SUT. You will not be debugging, changing, or editing anything in the webapp.

   Your first task is to study the web-app requirements in the appendix and use your browser to explore the web-app behavior until you are certain you understand what the web service is supposed to do.
   (Keep in mind, the web-app is known to have bugs).

2. Make a list of tests you think you should create in order to achieve acceptable test coverage for this webapp. What are the input values you want to use? What sequence of requests/operations do you want to test? How much of the returned form do you want to validate? This analysis should result in a test plan and test set. Document both: they will be deliverables for this lab.

III. Compose the tests and execute your test plan.

1. Create a simple java application project. Add a test source folder, as usual. Create a junit class (in the test folder) and start writing the tests required by your plan. Make certain that each test is named so that it is painfully obvious which test in the test set it implements.
   The web service is known to have at least two bugs.
2. Write defect reports for any bugs you find. Move the failing tests that reveal those bugs into a separate test class named Defects. Your successful tests, defect reports, and failing tests will all be deliverables for this lab.

Submission Instructions

When your work is done, you should place have deliverables in your project:

content	**name and location in Project**
Junit/Http tests which pass	test/TestLab5.java
Junit/Http tests which fail (defects detected)	test/Lab5Defects.java
Test Plan	doc/TestPlan.doc or doc/TestPlan.pdf
Description of Test set	doc/TestSet.doc or doc/TestSet.pdf

Please pay attention to the name and location specifics. The .java files should be in package *edu.iastate.cs417.lab5.*

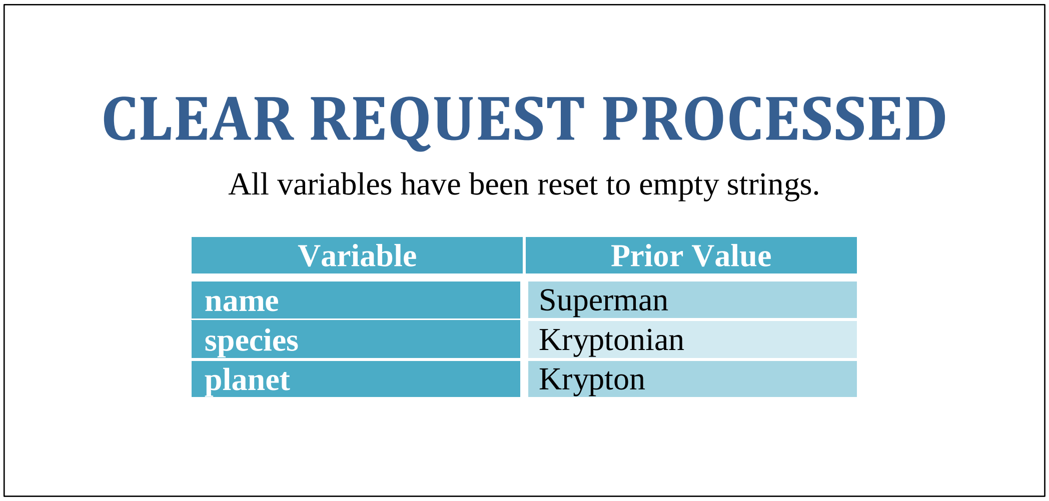
Package the project as a zip file, omitting binary class files and library jars. The resulting project jar should be quite small. If it isn’t, get some help from someone and remove the unwanted files before submitting.

We will take off points if we need to reorganize files or supply missing files or deal with unnecessarily large zips. Name the zip “<last name>_<first name>_417lab5.zip”. Submit via BlackBoard.

We will accept projects that are less than 24 hours late, with a 10 percent (of possible points) penalty.

Appendix 1 – lab 5 web service Requirements

1. The webapp needs is packaged in a war intended to be deployed at localhost:8080, context root lab5host.
2. This webapp will have no database, but will use application scope attributes to retain state for three variables: name, species, planet. As a prototype, the web app is not certified to support multiple simultaneous users.
3. The webapp will implement these endpoints and behaviors:
   - /lab5host/command/clear.jsp
     A GET request to this resource will return a page that lists the current value in the three variables and then clears all three to an empty string. The return page should look something like this:



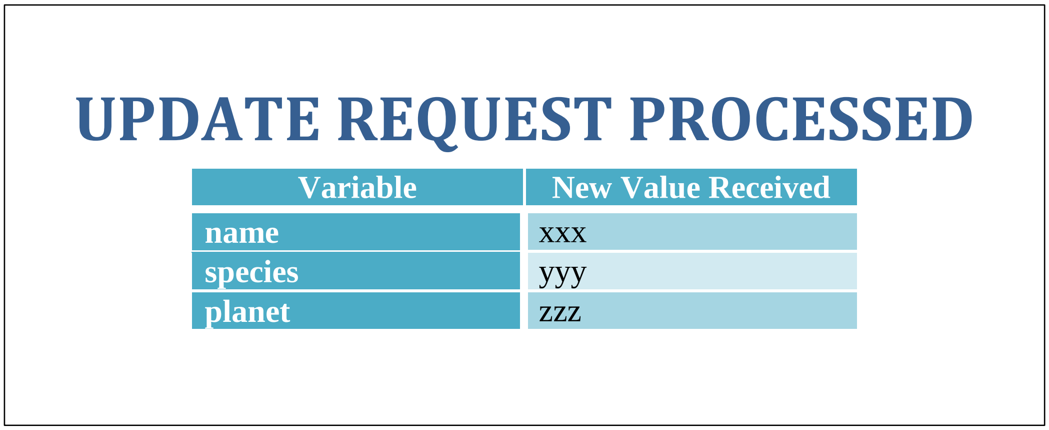
CLEAR REQUEST PROCESSED

All variables have been reset to empty strings.

Variable	Prior Value
name	Superman
species	Kryptonian
planet	Krypton

   - /lab5host/command/update.jsp
     A POST request to this resource will look for parameters “name”, “species”, and “planet”. If any of the three parameters do not exist *in the body of the request* or have null or all whitespace values, the webapp will return status code 422.
     A GET request will result in a 404 error.

Otherwise the endpoint will return a page that looks like this:



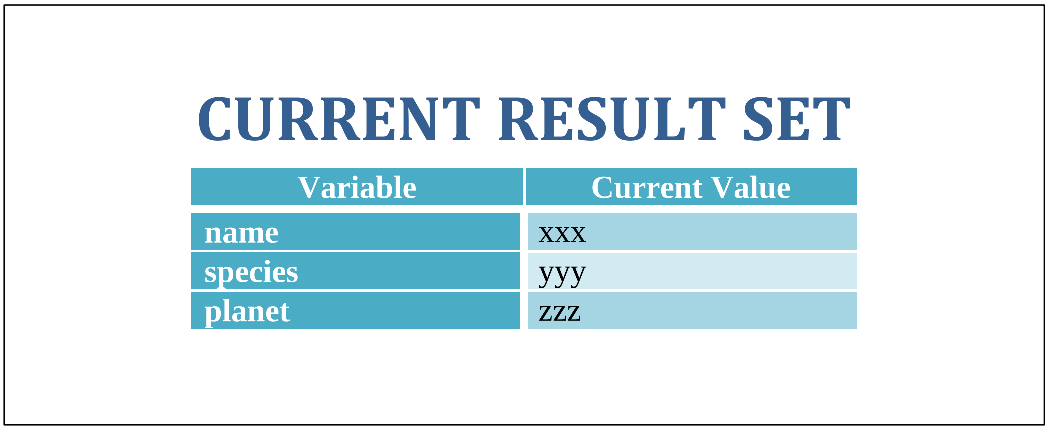
UPDATE REQUEST PROCESSED

Variable	New Value Received
name	xxx
species	yyy
planet	zzz

- /lab5host/query?planet=xxx
  This endpoint responds to GET requests only. If the value supplied for planet is the current value of planet, the webapp will return a string token in the response header “planet”.

  If the value supplied in the request for parameter planet is “TestThisEndpoint”, the response should be status code 200, but with no planet header.
  Otherwise this endpoint should return status code 404.
  (Testers can use this feature to confirm that they are reaching the endpoint, even if they aren’t supplying the correct information to get a token.)

- /lab5host/resultset?token=kkkk
  A GET request (only) for this endpoint with the last issued token value will return a page of the form:



CURRENT RESULT SET

Variable	Current Value
name	xxx
species	yyy
planet	zzz

If the request is other than GET or does *not* contain the most recently issued token, the endpoint will return an empty body and status code 404.

4. The production war file should use standard (web.xml) configuration so that it will deploy trivially in both glassfish and tomcat.

Teaching Objective

Normal usage scenario:

```
fetch Get clear.jsp to clear state.
one or more times
    update variables (post to update.jsp)
    zero or more times
            fetch a token for a particular planet. (get to query.jsp)
            use the token to retrieve the result set for that planet. (get to resultset.jsp)
    end loop
end outer loop
```