

SCHOOL OF COMPUTER SCIENCE



PRIFYSGOL
BANGOR
UNIVERSITY

Java Technologies

Mini-Project 4

Java Servlets

Dave Perkins

Introduction

In today's laboratory session we introduce the topic of Java server-side web technologies, beginning with Java Servlets. You will come to appreciate servlets as a highly-scalable & powerful technology suitable for enterprise-grade web applications. In the next two weeks, you will learn how to:

- create and deploy a servlet in a web application server such as Glassfish or Tomcat;
- use a servlet to generate XHTML pages;
- write servlets to respond to HTTP get and post requests;
- manipulate persistent data;
- use enumerations to manage form data;
- develop simple three-tier applications.

What is a Servlet?

The official answer to this question is provided in the J2EE tutorials provided by Oracle.

A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.

The `javax.servlet` and `javax.servlet.http` packages provide interfaces and classes for writing servlets. All servlets must implement the `Servlet` interface, which defines life-cycle methods.

When implementing a generic service, you can use or extend the `GenericServlet` class provided with the Java Servlet API. The `HttpServlet` class provides methods, such as `doGet` and `doPost`, for handling HTTP-specific services.

http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets2.html

These laboratory notes focus on writing servlets that generate responses to HTTP requests such as GET and POST. Accordingly, some knowledge of the HTTP protocol is assumed. If you are unfamiliar with this protocol, you can get a brief introduction to HTTP in Appendix A at the site quoted above.

Getting Started

The topic of Java Servlets has not been covered in a lecture course you should therefore refer to one or more of the following sources of information to get started:

- Slides posted on Blackboard
- Go to the **Useful Links** section in these notes and look at one or more of the on-line sources listed

To run a Java servlet you will need to have an application server such as Glassfish installed on the machine you are using. This should not be a problem as the machines in 3.19 should already have this piece of software installed.

Web application servers such as Glassfish and Tomcat are easily installed in the Netbeans environment, and these laboratory notes assume that this has already been done. For your information I developed solutions to these exercises using NetBeans 6.9 and Glassfish v3.

Once a server has been installed you can use the address bar in your web browser to test whether it is listening on port 8080. Simply type:

<http://localhost:8080/>

You should then get the Glassfish welcome page. Similarly for Tomcat.

For information about the Glassfish Enterprise Server consult the Quickstart Guide available at

<http://glassfish.java.net/docs/3.1.1/quick-start-guide.pdf>

If this link does not work please let me know as soon as possible.

Exercise 1: Building a Web Application in Maven (10%)

Using a current version of Netbeans (these notes were developed using 8.0.1) create a Maven Web Application. Make sure that you select **Web Application** as the project type.

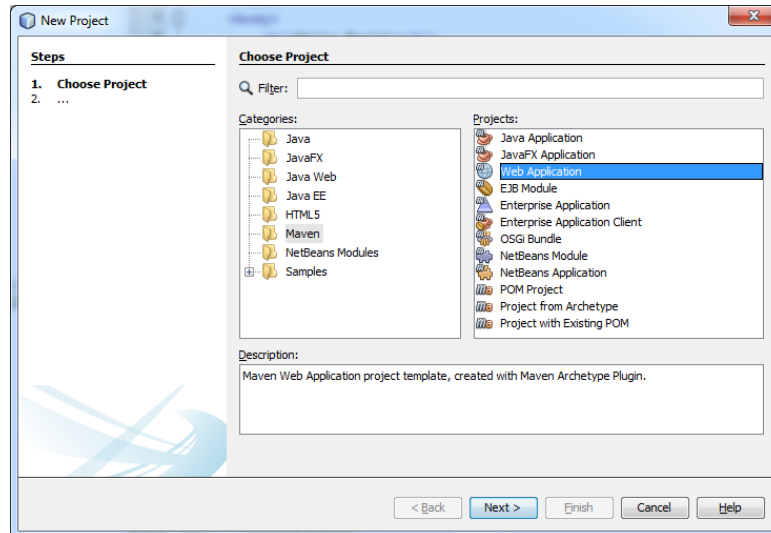


Figure 1 Creating a Maven Web Application

When Maven creates a web application a standard directory structure is set up and a number of files are automatically generated and stored in a **.WAR** file.

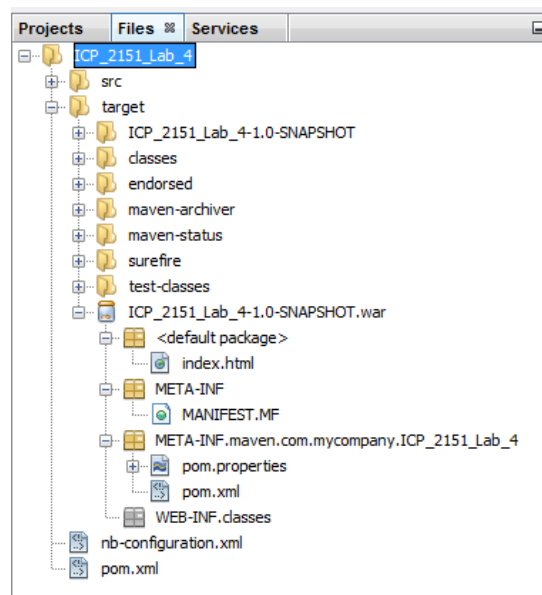


Figure 2 Project Structure of a Web Application

At this stage the most important file to locate is the **index.html** file which contains a short piece of html script. This file is automatically displayed whenever a browser is pointed to appropriate address.

As you can see this action results in the “Hello World!” message being displayed.

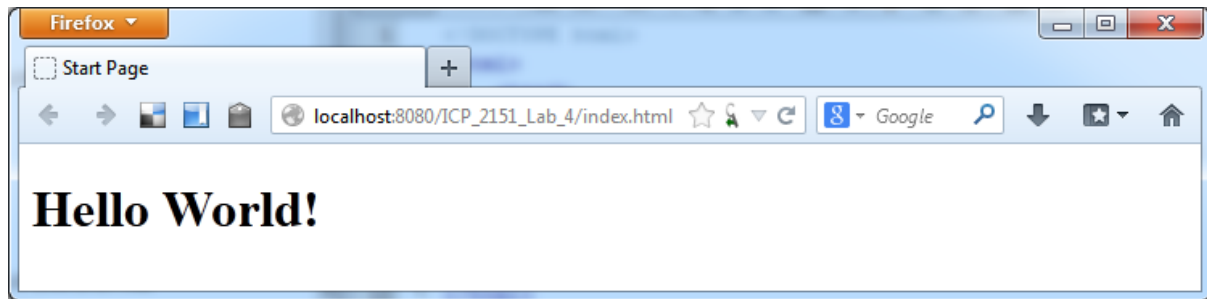


Figure 1 Displaying index.html

To run your web application you should right click on the blue **Web Project** icon and select the **Run** option. Note that this option results in the following sequence of events:

- Starts the Glassfish Web Server (if not already started)
- Automatically invokes Firefox
- Points the browser at the appropriate page

If everything has been done correctly the message should be displayed in the browser's window. This is illustrated in Fig 3 above.

Note also that you can obtain the same result independently of Netbeans by starting Glassfish from the shell window, then starting the browser and entering the URL below into the address box.

`http://localhost:8080/ICP_2151_Lab_4`

To complete this exercise, modify the file **index.html** so that the following sequence of messages is displayed:

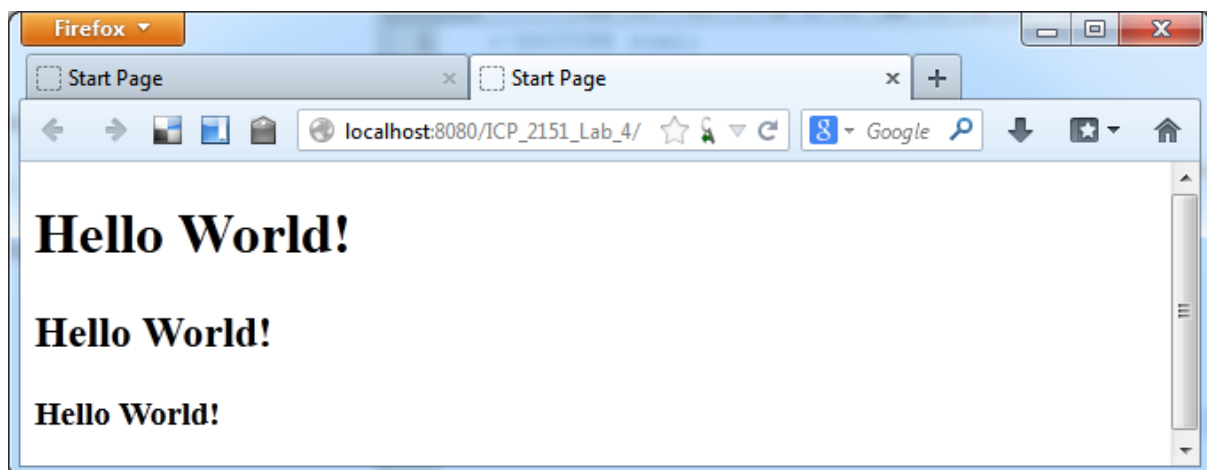


Figure 2 Modified Version of index.jsp

To obtain the above you need only insert two extra lines with appropriate HTML tags.

Exercise 2: Responding to a GET Request (10%)

The first servlet you are asked to write responds to a HTTP **GET** request for a specified URL. The **GET** request is generated by a user clicking a button on a web page. The sequence of events is illustrated below:

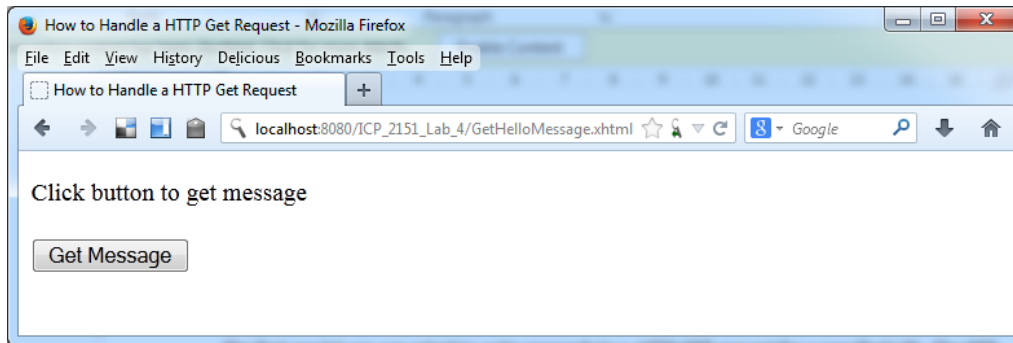


Figure 3 XHTML Document To Invoke a Servlet

When the user clicks the button labelled **Get Message** a servlet called **HelloWorldServlet** is invoked.

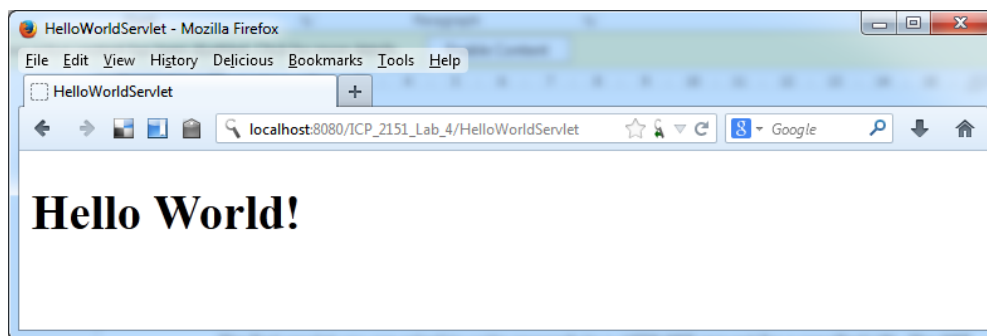


Figure 4 Message Displayed by HelloWorldServlet

The screen shot above was generated by running the web application from within the Netbeans IDE, however, the same effect can be obtained by directly entering the URL below in the address box of the web browser:

`http://localhost:8080/ICP_2151_Lab_4/HelloWorldServlet`

To help you develop a solution for this exercise I suggest that you follow the plan below

- Create a XHTML file called **GetHelloMessage.xhtml**
- Test this independently by right clicking on the file and using the **View** option
- Create a Java Servlet called **HelloWorldServlet** with a single method called **doGet** to handle the client request
- Right click on **GetHelloMessage.html** and this time select the **Run** option

For your information the contents of the project ICP_2151_Lab_4 is shown below. The files used in the current exercise are circled in red.

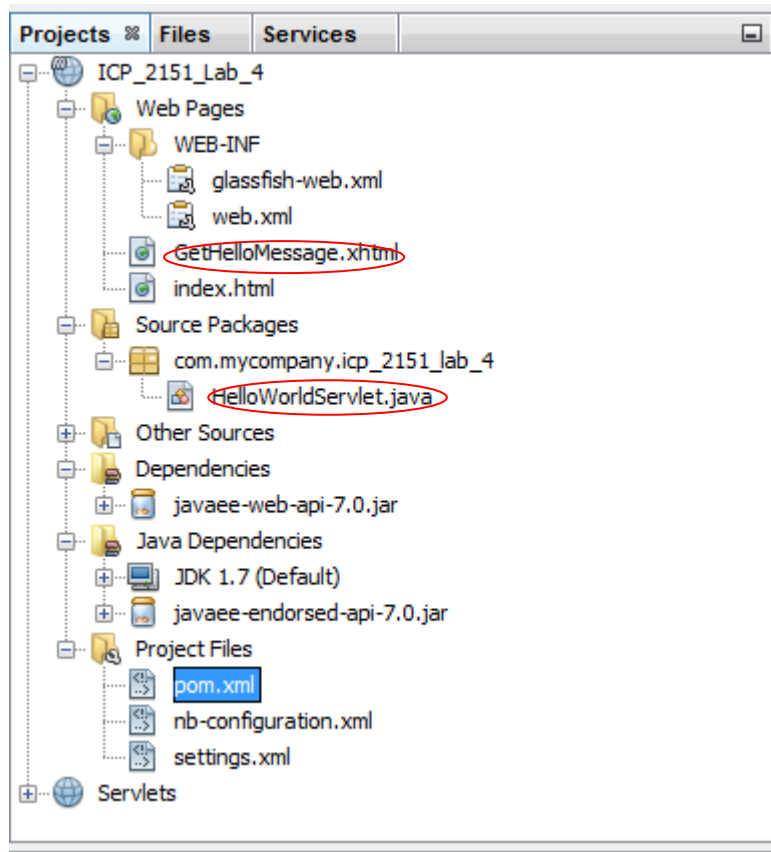


Figure 5 Project Structure

Finally before moving on to the next exercise you should look at the contents of the **WEB-INF** folder. In particular, look again at the file called **web.xml** which contains the *deployment descriptors* used by the servlet container to locate the various resources which make up a specific web application.

Exercise 3: HTTP GET Requests with Data(10%)

Write a servlet called **PersonalHelloServlet** which responds to a request containing data supplied by the user. For example, when the user clicks the **GetMessage** button the name supplied will be appended to the URL by the browser. This means that the address box will contain the following:

http://localhost:8080/ICP_2151_Lab_4/PersonalHelloServlet?name=Steve

The **?** acts as a separator between the name-value pair (i.e. **name=steve**) and the rest of the URL.

The operation of the servlet is illustrated in the screen shots below. The first screen shows the form generated by the file **GetPersonalMessage.xhtml**.

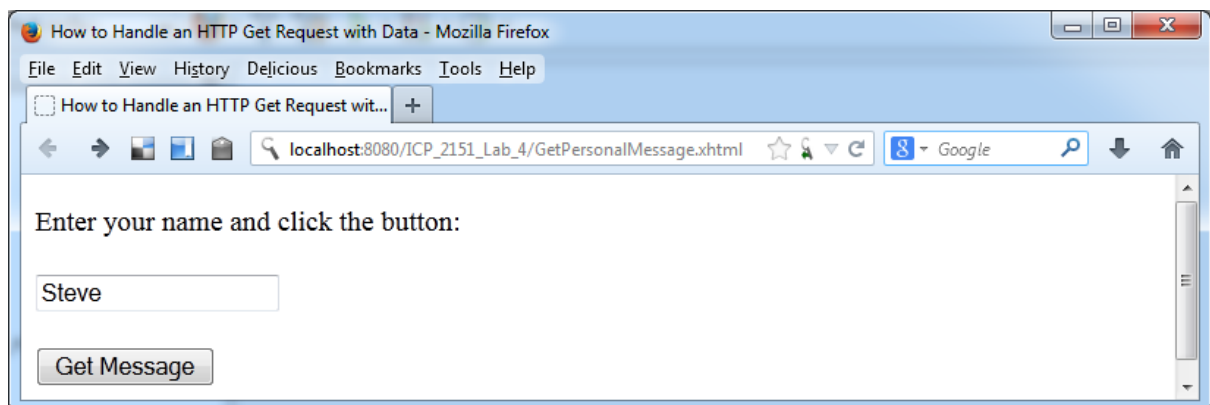


Figure 6 Supplying Form Data

When the user supplies a name and clicks on the button marked **Get Message** the a servlet is invoked and responds by producing the following page:

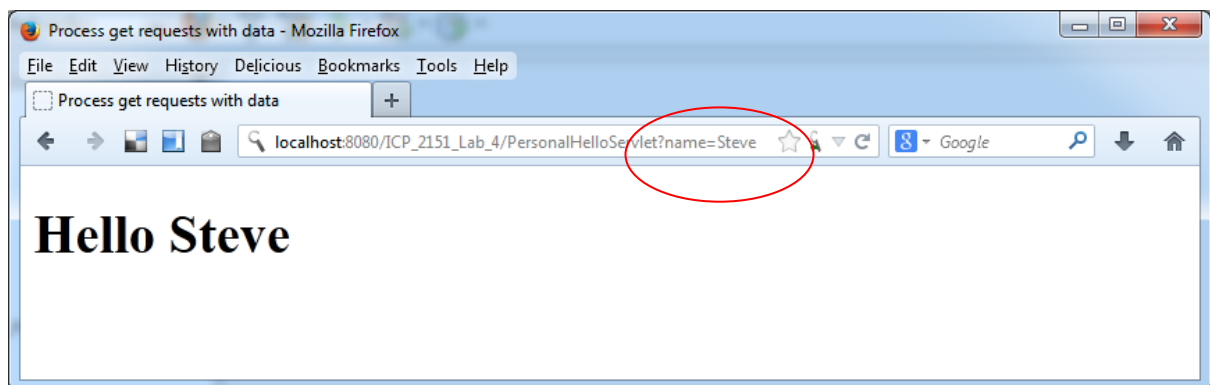


Figure 7 Servlet Uses Form Data to Generate Message

Notice how the name-value pair **name=Steve** appears in the address box of the browser.

Remember that to get your program working you will need to make additional entries to the **web.xml** file. In particular you will need a new **<servlet>** and **<servlet-mapping>** for the new servlet class you have created for this exercise.

Once your servlet can generate an appropriate message modify the code so that if the user does *not* enter a name a warning is issued.

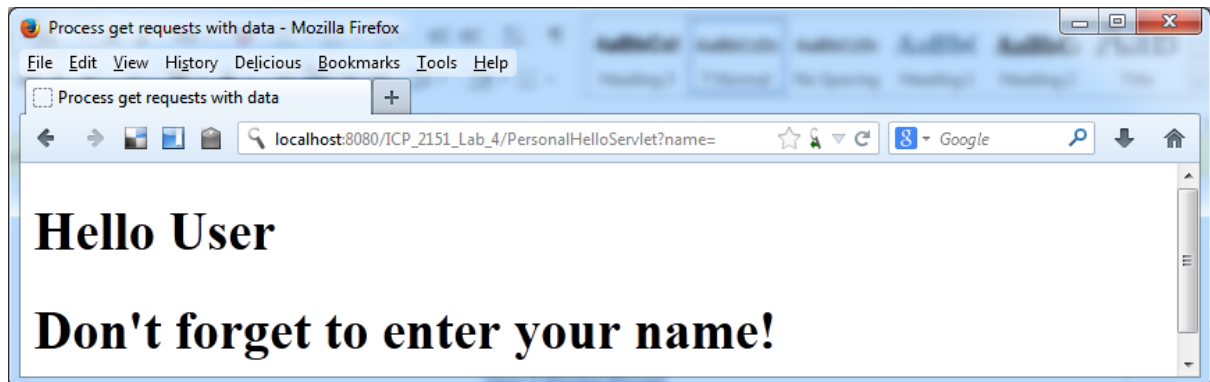


Figure 8 Warning Message

Hint.

To access form data (i.e. the name supplied) use the method **getParameter ()** which is defined in the class **ServletRequest**.

Exercise 4: Handling HTTP POST Requests (10%)

For this exercise you are asked to create a web application with a standard login form. The login form will be created as a web page - using a file called **LoginForm.xhtml**. This form posts data to a servlet called **LoginServlet.java**. The form should allow the user to first enter a user name and a password, the server will then respond by issuing a personalised welcome message to the user.

Characters typed in the password text box must *not* be echoed.

In the XHTML file place the following form tag:

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

```
..
```

```
</form>
```

This form will ensure that the web browser submits an HTTP **post** request to the servlet rather than an HTTP **get** request.

Test that when your XHTML file is rendered by the browser you get something like the following.

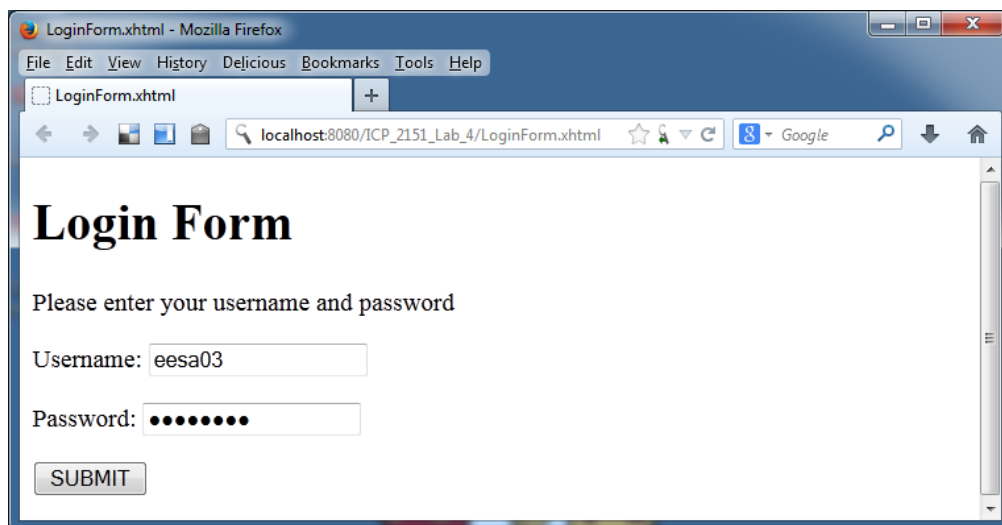


Figure 9 Login Form

For the next part of this exercise use the **getParameter()** method to extract information from the **HttpServletRequest** parameter in **LoginServlet.java**; in particular you will need to obtain the user name in order to send a personalised greeting message back to the client.

The servlet **LoginServlet** which you are asked to produce should generate output similar to the following. Note that to complete this exercise correctly you *must* display the user name.

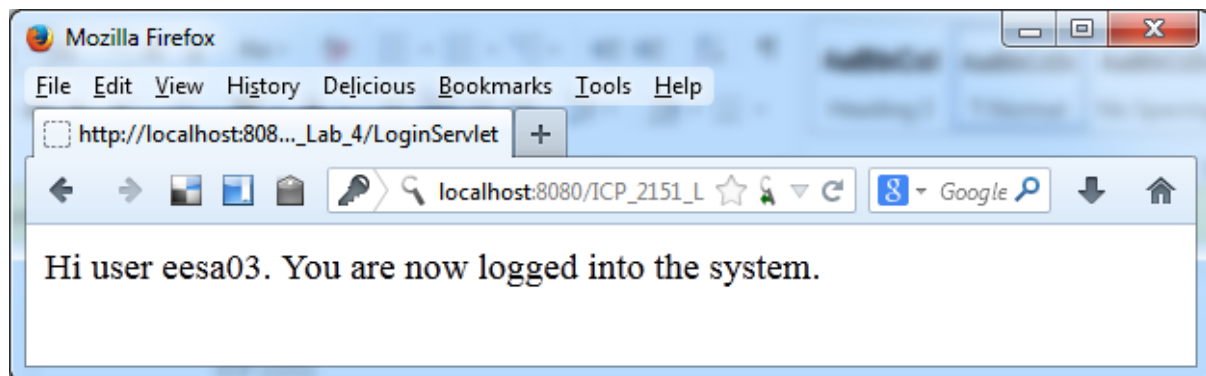
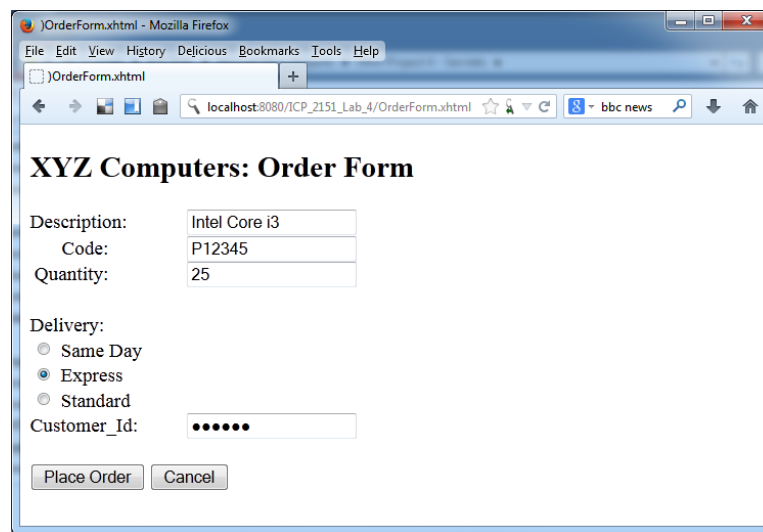


Figure 10 Servlet Response to Client

Finally, remember to modify your web.xml file and include appropriate tags for your new servlet.

Exercise 5: Managing Forms with an Enumeration (15%)

Create a XHTML page to display the form below. Bear in mind there are some marks for layout!



XYZ Computers: Order Form

Description:

Code:

Quantity:

Delivery:

☐ Same Day

☒ Express

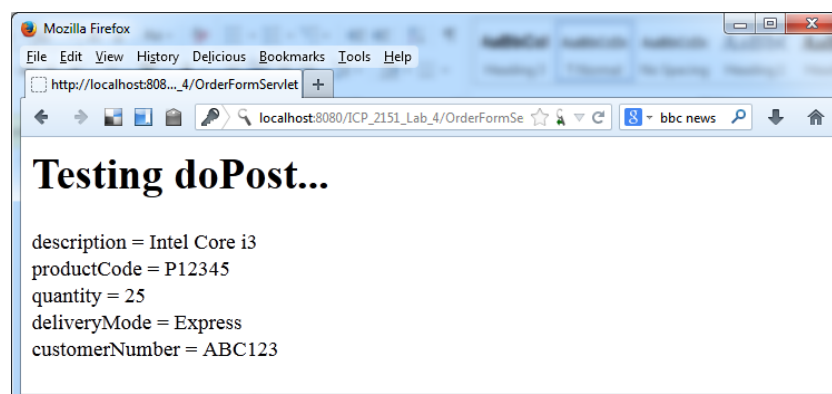
☐ Standard

Customer_Id:

Figure 11 OrderForm.html

Implement a servlet **OrderFormServlet** which uses the **request** object (i.e. the first parameter of a **doXXX** method) to recover all the information contained in the form data. Your servlet should contain a **doPost** method which displays the name and value of each parameter supplied via the **request** object. To manage the various parameters holding the form data items you should use an instance of the class **java.util.Enumeration**.

For test purposes a page similar to the one below should be returned to the client.



Testing doPost...

description = Intel Core i3

productCode = P12345

quantity = 25

deliveryMode = Express

customerNumber = ABC123

Figure 12 Testing doPost

If you are having trouble laying out your form, have a look in the [Useful Links](#) section in these notes.

Exercise 6: Building a Hit Counter (15%)

Write a servlet called **RandomFactServlet** which generates random facts on request. The servlet also maintains a hit counter such that each invocation of the servlet's **doGet** method results in the hit counter being incremented.

The initial XHTML page is shown below:

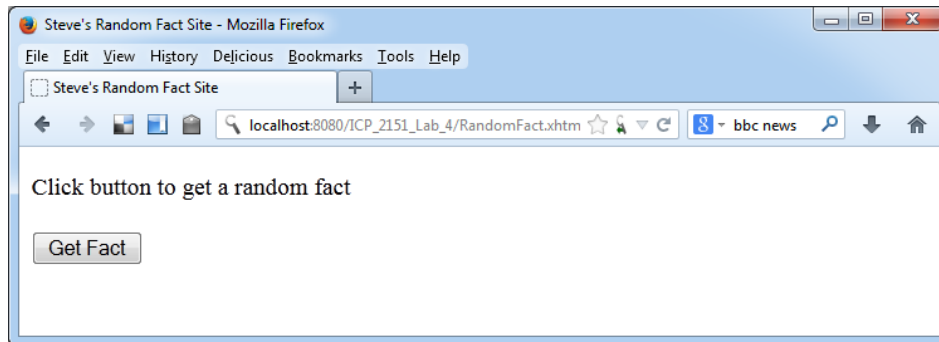


Figure 13 Initial Page

Once the **Get Fact** button has been clicked **RandomFactServlet** will respond by generating a page similar to the one below. Notice the hit counter on the bottom left of the page.

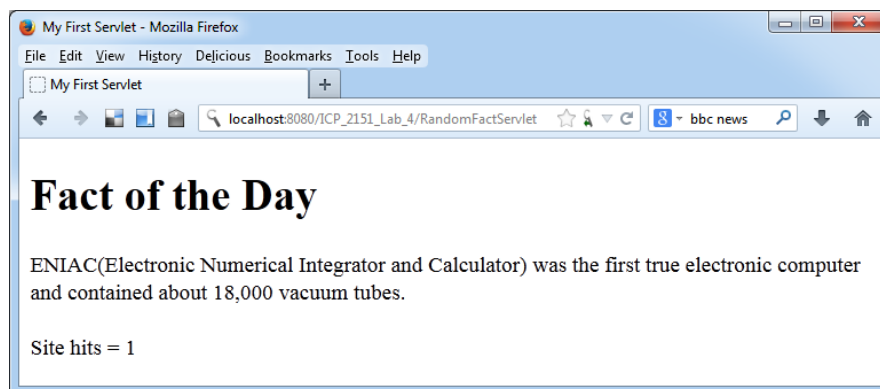


Figure 14 Displaying a Random Fact

For purposes of this assessment I suggest you construct a text file containing ten or so random facts about the history of computing - an interesting subject in its own right and about which far too many students know too little!

Exercise 7: A Time Zone Web Applications (30%)

Write a web application to look up the current time in a given time zone. The application begins by presenting the user with the following page:

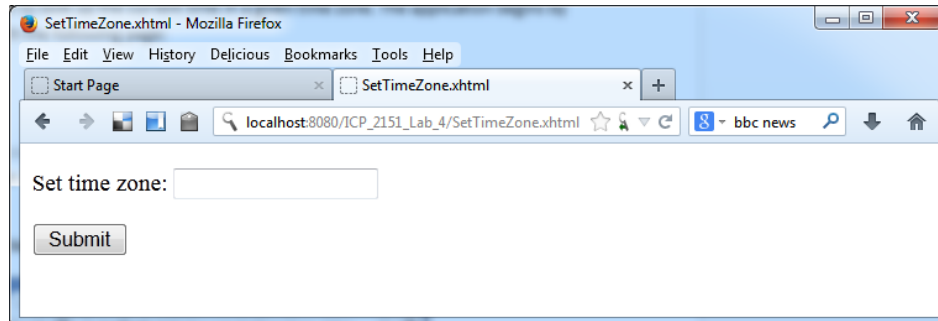


Figure 15 Initial Page

So for example, when I wrote this sentence it was exactly 10:17:18(GMT) in Bangor, Wales but in Paris, France the time was 11:18:21. Your application should provide this kind of time zone information. The application works like this: first enter the city to specify the time zone:

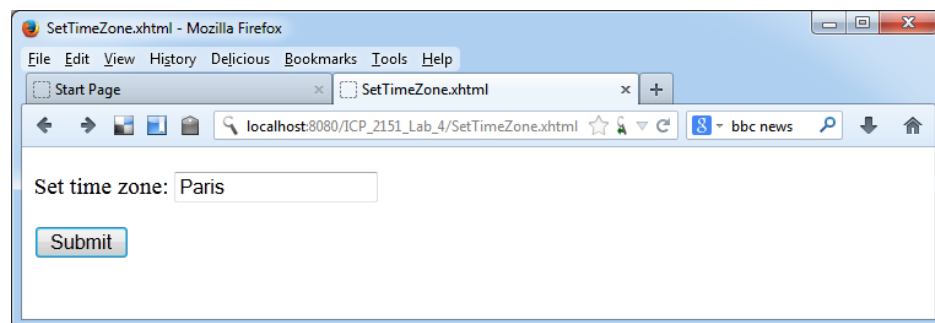


Figure 16 Setting a Time Zone

When the **Submit** button is clicked the following response is generated:

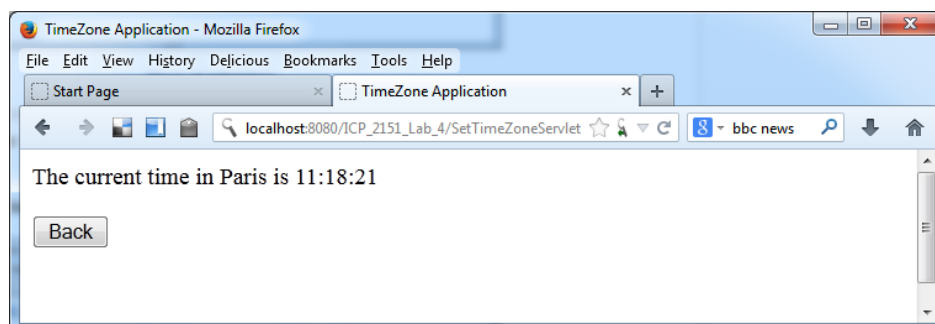


Figure 17 Paris Time Displayed

Horstmann's *Big Java* contains a version of this program (see Chapter 23) but it is written using *Java Server Faces* technology. However, it is worth studying Horstmann's solution so as to acquire information about what Java library support exists for time zone programming.

Hint. Look up details of the Java library class **TimeZone**.

Your program should have a degree of error handling. Thus if the user supplies a fictitious city or indeed a city which the **TimeZone** class does not know about then an error message should appear. The page flow is illustrated below.

User supplies name of a non-existent city

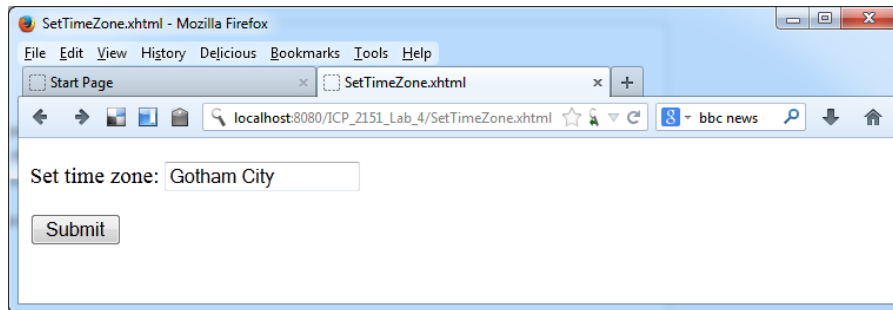


Figure 18 Fictitious City

Application catches this error and notifies the user.

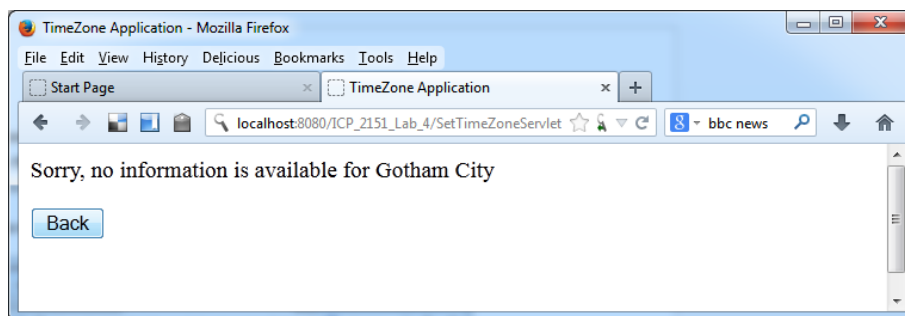


Figure 19 Error Message

When the user clicks the **Back** button they are taken back to the page illustrated in **Figure 19**.

This exercise involves the following topics:

- Servlet interaction with other Java classes managing the underlying data model
- Page navigation
- Error handling

Useful Links

The links below contain plenty of information about servlets and how to use them.

- Wikipedia - "Java Servlet" – http://en.wikipedia.org/wiki/Java_Servlet
- The site below contains a legal .PDF version of an excellent introductory book *Core Servlets and Java Server Pages* by Marty Hall and Larry Brown.
<http://pdf.coreservlets.com/>
- Oracle provides a basic introductory tutorial at
<http://www.oracle.com/technetwork/java/servlet-142430.html>
- Oracle also provides some more advanced on-line tutorials at these two sites:
<http://java.sun.com/developer/onlineTraining/Servlets/Fundamentals/introduction.html>
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html
- Other useful introductory sites are to be found at
<http://www.novocode.com/doc/servlet-essentials/>
<http://www.servletworld.com/servlet-tutorials/index.html>
<http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/Servlet-Tutorial-First-Servlets.html>
- Some tips for form layout using CSS can be found at
http://www.dailycoding.com/Posts/layout_form_without_tables_with_css_trick.aspx

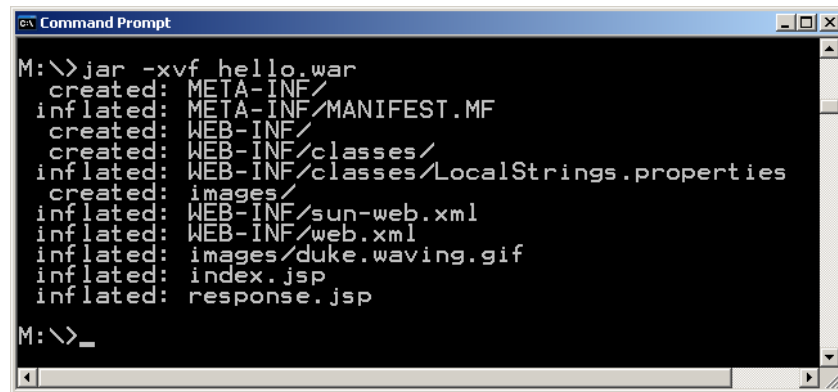
If you find anything particularly useful do let me know.

Appendix 1: What is in a .WAR File?

For those who are curious a **.WAR** file can be opened just like a **.JAR** file from the command line by using the **jar** program:

```
M:\> jar -xvf hello.war
```

Assuming everything is in place you should get the following listing:



```
Command Prompt
M:\>jar -xvf hello.war
created: META-INF/
inflated: META-INF/MANIFEST.MF
created: WEB-INF/
created: WEB-INF/classes/
inflated: WEB-INF/classes/LocalStrings.properties
created: images/
inflated: WEB-INF/sun-web.xml
inflated: WEB-INF/web.xml
inflated: images/duke.waving.gif
inflated: index.jsp
inflated: response.jsp
M:\>_
```

Figure 20 Contents of hello.war

Notice that the directory structure of the **.WAR** file is reproduced in the **domain1** folder which is in the Glassfish server.