# Servlets & JSP

Lab Guide

12824-0000-04

JOHN BRYCE
Leading in IT Education
matrix company

# Servlets & JSP

## Lab Guide

Written By: Rony Keren

Internet Team

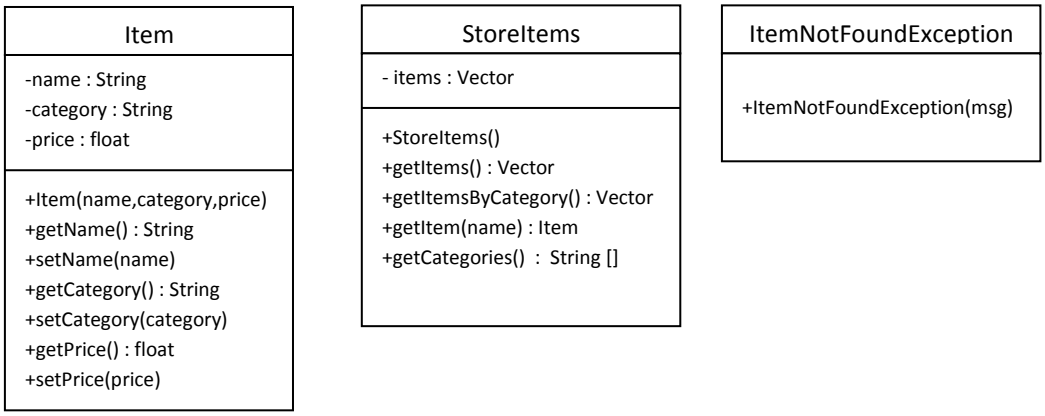
John Bryce Training LTD

# Servlets

## Lab 1 – Create A Dynamic Web Project

In order to complete this exercise follow these steps:

1. Open EclipseWTP IDE
2. Create a new web project
   - in the new project wizard go to 'Web'
   - choose 'Dynamic Web Project' and press 'Next'
   - in the next screen set your project name to 'OnlineStore'
   - verify or set the runtime target to use Tomcat
   - press finish
   - if Eclipse offers to change to the JEE perspective – approve
3. Expand the 'OnlineStore' web project in the package explorer view
4. Expand the WebContent node. You should see that a WEB-INF directory was already created
5. Expand the WEB-INF directory. You should see that a 'lib' directory is located in it
6. Drag db.jar file located under Labs\lab1\lib to the project lib directory
   - this jar contains classes that are used as the store DB
     - Item – represents an item that can be added to a cart
     - StoreItems – contains all available items in the store




   - dragging the jar to your project lib directory makes it available to all server side classes included in your web module (Servlets, JSPs and helper classes)
7. Examine the project infrastructure
   - Project name is the root context of the web module
   - server side classes (Servlets, beans & helper classes) should be located under 'src' node. Compilations pasted to WEB-INF\classes when packed and\or deployed to the server automatically.
   - JSPs and HTML files are created under WebContent and packed under root context when packed and\or deployed automatically.

## Lab 2 – Controller Servlet

In this lab you will create a servlet. This servlet is designated to do most of the 'business logic' implemented in Java rather than HTML.

Basically, this servlet gets commands from submitted pages and decides how it should be handled. It is also decides which page will be used for returning results. Therefore, it is usually referred as 'Controller servlet'. It is part of the MVC architecture explained in details later on during this course.

To complete this lab follow the next steps:

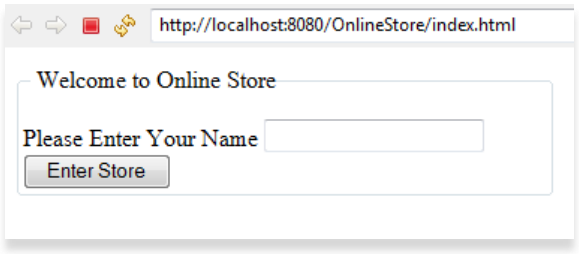
1. Right click the project node in package explorer and choose New -> Servlet
    - If Servlet is not shown you should make sure that you are using 'JEE Perspective' but you can always go to New -> Other -> Web -> Servlet
2. Set class name to be 'store.Controller' and click 'Next'
3. Add an initial parameter: name – 'email', value – 'support@online.store'
4. Verify that URL Mapping is set to /Controller
5. Click 'Finish'
    - Now you should see that the java source in located under 'Java Resources :src' node, in a default package.
6. Edit the Controller servlet class
    - Define the following private attributes:
        - supportMail : String (holds the email predefined in web.xml)
        - hitCounter : int (count 'Buy' operations)
        - store : StoreItems (holds item list for sale)
    - Add init(ServletConfig config) method
        - instantiate store
        - load 'email' init parameter to set supportMail
    - Replace doGet() and doPost() with service() to respond to both GET & POST requests
7. If you want to actually see that this servlet is working simply print messages to screen in init() and service()
    - You should know by now that init() will print its message only on the first call to this servlet – when it is created. service(), on the other hand, should print a message each call.
8. You may also add a @PostConstruct method just to see how it is invoked right after init() – we will not need it in the following labs..
9. To run your servlet simply right click it on package explorer and select Run As-> Run On Server
    - Verify that you are using Tomcat 7.0 or up as your Runtime Target
    - Eclipse will use the URL pattern defined in web.xml to point to the Controller servlet: http://localhost:8080/OnlineStore/Controller

## Lab 3 – Creating Login Servlet & Sending User Name

In this lab you will create a servlet that manages login operation. Currently, it will only get user name or provide default name but later on it will set it automatically via cookies and a filter.

In order to complete this lab, do the following:

1. Create a servlet
    - name – LoginServlet
    - url pattern - /Login
    - Override the service() method
        - obtain a request parameter with the name 'userName' from HttpServletRequest
        - Print it to screen
2. Create an HTML file
    - Eclipse provides an HTML template
        - Right click your project node in the package explorer and select New -> HTML
        - When file is created, you may right click it on the package explorer and choose Open With-> Web Page Editor. This will open a graphic view and a palette. You may drag components directly to the page.
    - name – index.html
    - create a form
        - form action should be 'Login' – in order to point it to the LoginServlet
        - add an input field named 'userName'
        - add a submit button
    - HTML should look like that:




3. Run the HTML on server and verify that the submitted data is loaded and printed by LoginServlet successfully.

## Lab 4 – Writing data to the client

In this lab you will update login servlet so it will generate a welcome page. The servlet also loads user name to the request scope for future use.

In order to complete this lab, do the following:

1. Update LoginServlet
    - Load user name on request scope via request.setAttribute() method
        - attribute name – username
        - attribute value – entered by user or default ('Guest')
    - Return an HTML response that greets the client and uses its user name
        - set content type of the response to be text/html
        - the result should be very simple and look like that:


2. Optional: update index.html to submit the form using POST method
    - that way, the entered user name will not appear in the assigned URL when submitting the form
        - edit index.html <form> element
        - add attribute named 'method' with the value 'POST'
        - default is method is 'GET'

## Lab 5 – Adding and managing 'Last visit' cookie

In this lab you will update login servlet to handle last visit cookies. Last visit cookie holds the last date a client performed the login operation. The information will be presented to the client when logged in.

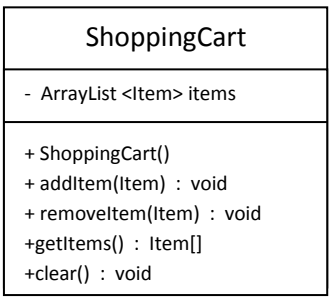
In order to complete this lab, do the following:

1. Update LoginServlet
    - Load cookies from the request object
        - verify that the cookie array in not null and has elements
    - Seek for previous planted visit cookie
        - if found – store its value in a variable to present it later
        - if not – set current date as the value to be presented
    - Add an updated visit cookie with current date to the response
    - Set cookie expiration time to one week
    - Print visit data – should look like that :

## Lab 6 – Managing shopping carts within sessions

In this lab you will create a class that manages shopping carts and bind it with HTTP sessions. In order to do that you'll add two additional commands to the controller. You will also make sure that existing shopping carts are invalidated when clients perform log-in operation.
In order to complete this lab, do the following:

1. Create ShoppingCart class that holds all client items and is bounded to the session context
    - Create a package named 'helpers' under project src node
    - Create a class named 'ShoppingCart' that follows this outline:
        - A template source is provided in Labs\lab6\src directory




    - Implement all methods to add, remove, return and clear items accordingly
        - Item class is packed in db.jar you added to your project lib directory
2. Edit Controller servlet – service() method
    - Get the command parameter from the request and store it in a String variable named 'command'
    - Check command value. If it is equals to "startShopping"
        - create a session
        - verify it is really a new session
        - if it is a new session – create a new ShoppingCart instance & add it to the session as attribute named 'cart'
    - Add another condition to check command value. If it is equals to "clear"
        - get session (don't create a new one !) from the request object
        - get 'cart' attribute that should return a ShoppingCart instance
        - clear the cart
3. Edit LoginServlet to invalidate sessions whenever client logs in. Since clients may use 'Back' and 'Next' browser buttons they might send requests in a non-logical order – for example: a client can login, add some items to his cart and then login again. In this case we would like to invalidate both session and shopping cart so client will have a brand new session every time he logs in.
    - Get a session from the request (don't create a new one !)
    - Check if it is not null and if it doesn't – invalidate it

## Lab 7 – Working with ServletContext

In this lab you will load StoreItems instance in the web module context. You will also use ServletContext to forward requests to a View Manager servlet. View Managers are responsible for delegating requests, after processing, to the next page that will present response.

In order to complete this lab, do the following:

1. Create a new servlet named ViewManager
    - Override service to simply get the 'command' parameter from the request and print it to the screen
2. Update the Controller servlet
    - Update init() method
        - Call super.init(config) to get the context populated
        - Delete the line that instantiates a StoreItems object
        - Load the servlet context
        - Store an attribute in the context named 'storeItems' bounded with a new StoreItems object
        - This will assure that all units in this current web module are using a single instance of StoreItems which is a 'read only' object.
    - Update service() method
        - at the end of it, forward the request to the ViewManager servlet
        - use "/ViewManager" mapping name
        - Means that after the controller processed client request and loaded all the information needed for generating response – it will let the view manager to decide which page will be used to present that response. Meanwhile, we have no pages and the view manager simply prints the command to the default output. This helps you verify that forward operation actually performed.
    - In order to verify that dispatching occurs – invoke the controller with a dummy command:
      http://localhost:8080/OnlineStore/Controller?command=dummy

## Lab 8 – Combining Filters

In this lab you will create a Session-Filter and bind it to the controller chain. SessionFilter should block any operation targeted to shopping cart whenever session is no longer valid. Think of a scenario in which a client chooses several items and then remains inactive for an hour. In this case, client session is timed out (default time to live is 30 minutes) and therefore invalidated automatically by the server. The client is not aware of this and now tries to add another item. The SessionFilter suppose to track this situation and redirect the client back to the login page.

In order to complete this lab, do the following:

1. Create a SessionFilter class under 'helpers' package
    - Denote the class with @WebFilter annotation with name "sessionFilter"
    - Implement Filter interface
    - Map your filter to "/sessionFilter" url patterns via @FilterMapping annotation
    - Override the doFilter() method to perform redirection to http://localhost:8080/OnlineStore only if the session is null (invalidated) and the command is one of the following:
        - addItem
        - removeItem
        - clear
        - showItems
        - "" (empty string)
    - Leave destroy() and init() methods empty
    - Note: in order to load session and perform redirect you'll need to cast both request & response to HttpServletRequest & HttpServletResponse
2. Set mapping and bind with the controller in web.xml
    - Copy web.xml template file from lab8\resources to your project WEB-INF directory
    - Redefine SessionFilter in the web.xml via <filter> annotation – Tomcat doesn't support the annotation configuration yet..
    - Map it to the controller URL pattern ("/Controller") using web.xml using <filter-mapping>
    - You may check that requests are going through the filter by a simple print