Instructor Demos

Lab Exercises

EXERCISES/TUTORIALS

- Secure Java Web Application Development Lifecycle (SDL)

TUTORIAL: WORKING WITH ECLIPSE (JEE VERSION) AND TOMEE 7.X

Overview	
In this tutorial, you will: • Setup a Dynamic Web Project, import some resources, and run the web application on a TomEE instance. By the end of the lab you should be able to: • Setup a web project and deploy it to an instance of Apache TomEE • Verify that Apache TomEE is operating and determine its status through a variety of means	
Objective	To learn how to use Eclipse with TomEE
Builds on Previous Labs	None
Time to Complete	30 minutes

About

Apache TomEE is the Java Enterprise Edition of Apache Tomcat. Apache Tomcat 'only' provides a servlet container to support web based technologies like Servlets, JSPs and WebSockets. Apache TomEE adds the Java EE Web profile stack to the Tomcat base, adding support for technologies like CDI, JPA, EJB, JTA and more.

Apache TomEE+ adds even more technologies to the technology stack. TomEE++ also includes implementations for JAX-RS, JAX-WS and JMS.

Preface

During this training, you will be developing several mini-projects, using Eclipse and Apache TomEE. For each exercise, we have created skeleton code which you will be editing. In some cases, we also provide additional files which already have been implemented for you. For example, helper classes and some welcome pages for the web applications may be provided for you.

Before each exercise, all of these files (skeleton code, helper classes, etc…) will have to be imported into Eclipse. This tutorial describes how to work with various aspects of Eclipse (JEE Version) as well as Apache TomEE.

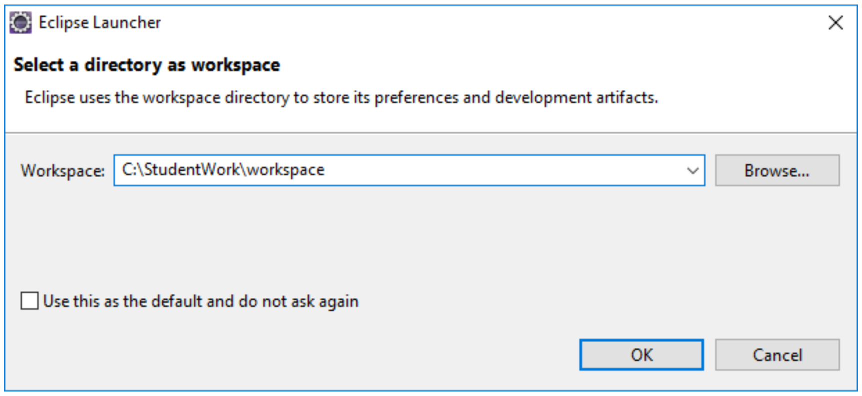
The Workspace Directory

During this tutorial, we will presume that the workspace directory has been placed in **C:\StudentWork**.

Starting Eclipse

1. From the Windows desktop, double-click the Eclipse icon. If you do not have a shortcut icon on your desktop, then add one that references the Eclipse executable.

   A popup will appear in which you can specify the location of the workspace.
2. Change the workspace location to: **~StudentWork\workspace**.


Eclipse Launcher

Select a directory as workspace

Eclipse uses the workspace directory to store its preferences and development artifacts.

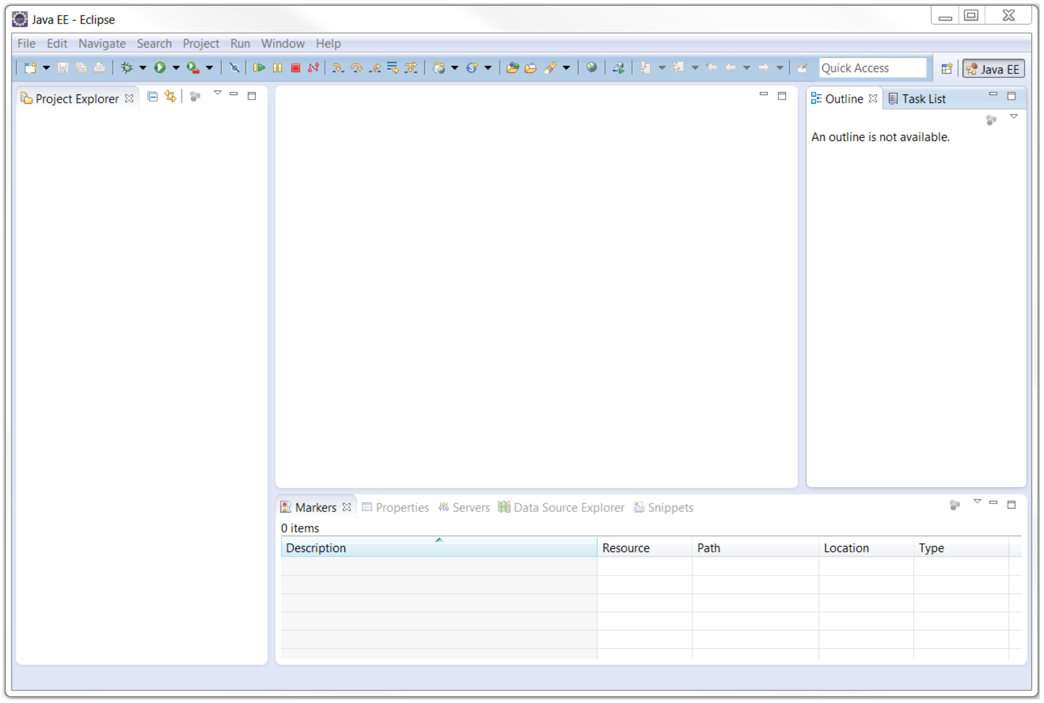
Workspace: C:\StudentWork\workspace

Use this as the default and do not ask again

An empty workspace will now be created in the **~StudentWork\workspace** directory.

When the application is started, the Welcome page will be presented.

Close the welcome screen, this will take you to the workbench:


Java EE - Eclipse

An outline is not available.

0 items

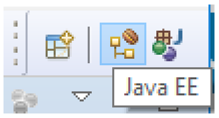
Description	Resource	Path	Location	Type

Configuring Eclipse (JEE Version) for TomEE 7.0.x

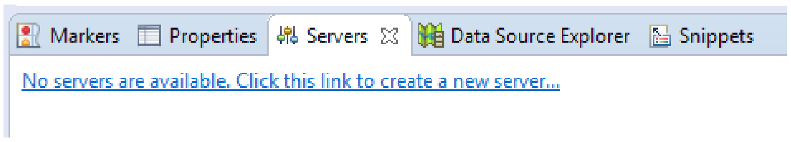
Eclipse needs to be configured to support whatever web server is being used in this class. The following instructions are specific to **Apache TomEE 7.x**.

The web server should already be installed on your local machine.

3. Make sure you are in the **Java EE perspective**

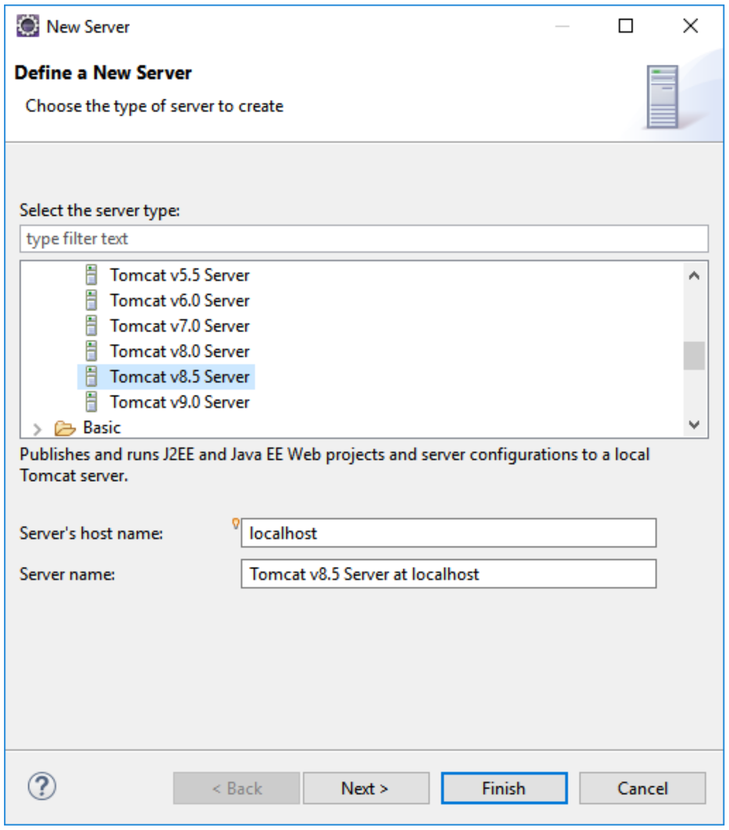


4. Navigate to the **Servers view** and click on the text that is displayed in this view.



No servers are available. Click this link to create a new server...

5. In the New Server window select **Apache -> Tomcat v8.5 Server** and click **Next**.


New Server

Define a New Server

Choose the type of server to create

Select the server type:

type filter text

- Tomcat v5.5 Server
- Tomcat v6.0 Server
- Tomcat v7.0 Server
- Tomcat v8.0 Server
- Tomcat v8.5 Server
- Tomcat v9.0 Server
- Basic

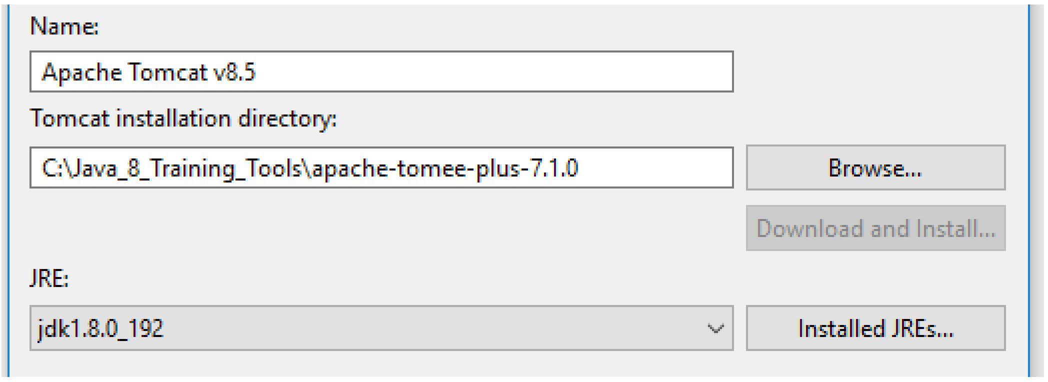
Publishes and runs J2EE and Java EE Web projects and server configurations to a local Tomcat server.

Server's host name:	localhost
Server name:	Tomcat v8.5 Server at localhost

Note: Even though we are using **Apache TomEE 7.x**, you will need to select the **Tomcat v8.5** server adapter!

6. Click **Browse** and navigate to the installation directory for TomEE.

7. From the dropdown-box select the full JDK and click **Finish**.


Name:

Apache Tomcat v8.5

Tomcat installation directory:

C:\Java_8_Training_Tools\apache-tomee-plus-7.1.0

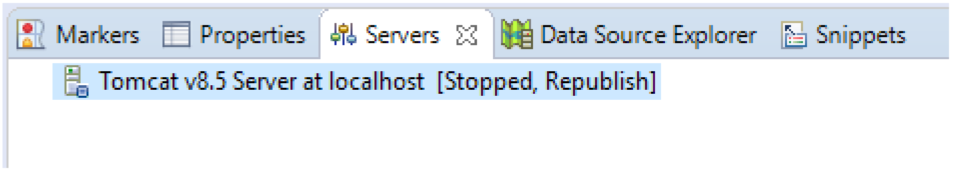
JRE:

jdk1.8.0_192

Configure the server instance

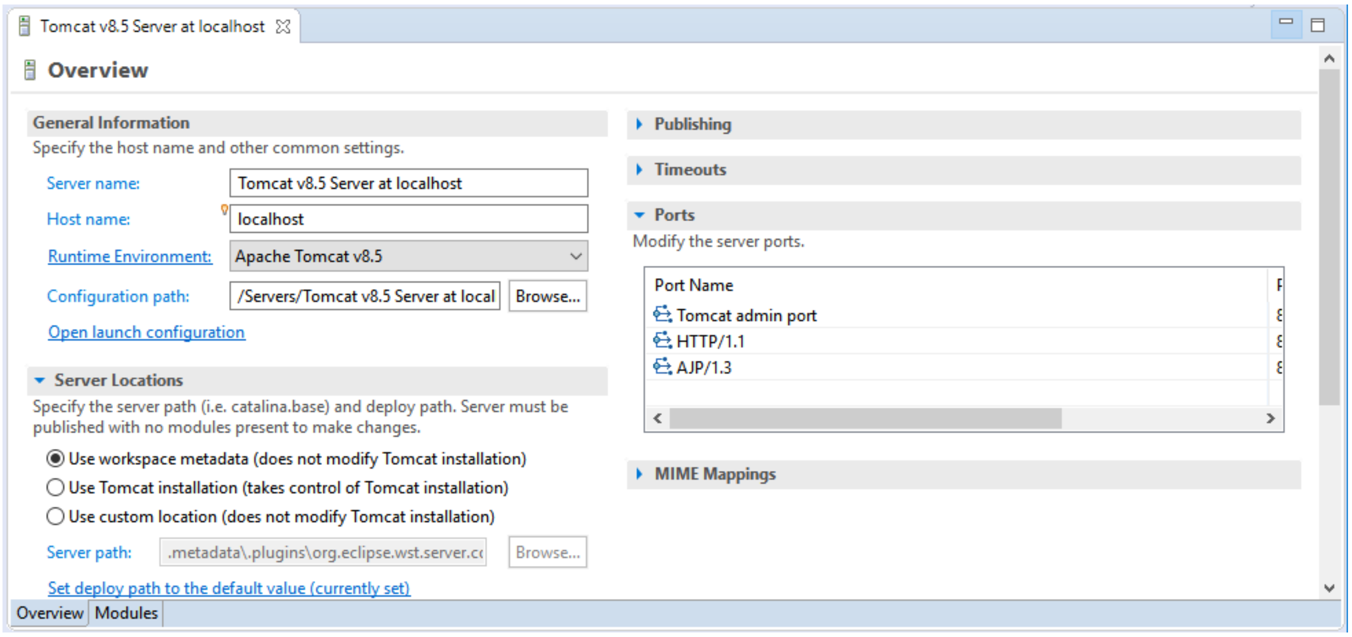
You will now go through the process of setting up a server instance, configuring it, and then running it to verify that everything is configured correctly.

8. Within the **Server** view you will see the entry for the new server that you have created. Note that it is currently stopped. Most operations that you want to perform with the server can be accessed by right-clicking on the server entry.



Tomcat v8.5 Server at localhost [Stopped, Republish]

9. Double-click on the server entry. This will open the Server Overview and show you various configuration entries for this server instance.

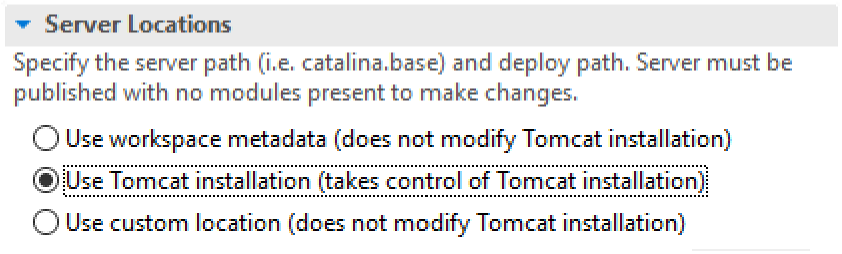

Tomcat v8.5 Server at localhost
Overview
General Information
Specify the host name and other common settings.
Publishing
Timeouts
Ports
Modify the server ports.
Port Name
Tomcat admin port
HTTP/1.1
AJP/1.3
MIME Mappings
Server Locations
Specify the server path (i.e. catalina.base) and deploy path. Server must be published with no modules present to make changes.
Open launch configuration
Set deploy path to the default value (currently set)
Browse...
Overview
Modules

Server name: Tomcat v8.5 Server at localhost
Host name: localhost
Runtime Environment: Apache Tomcat v8.5
Configuration path: /Servers/Tomcat v8.5 Server at local

[✓ Use workspace metadata (does not modify Tomcat installation)]
Use Tomcat installation (takes control of Tomcat installation)
Use custom location (does not modify Tomcat installation)

Server path: .metadata\.plugins\org.eclipse.wst.server.cc

10. Select **Use Tomcat installation (takes control of Tomcat installation)**.

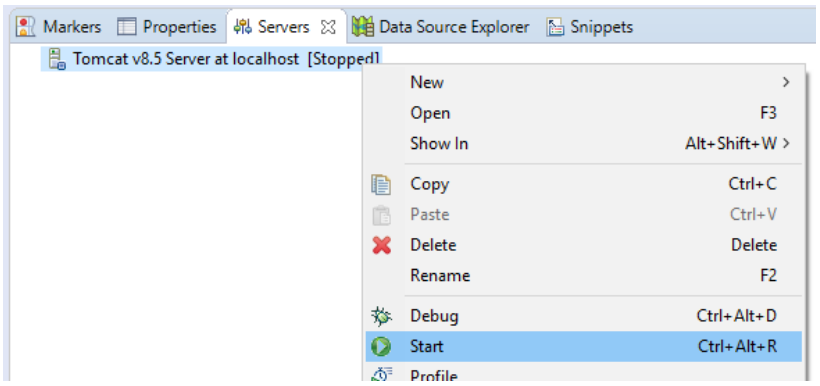

Server Locations
Specify the server path (i.e. catalina.base) and deploy path. Server must be published with no modules present to make changes.

Use workspace metadata (does not modify Tomcat installation)
[✓ Use Tomcat installation (takes control of Tomcat installation)]
Use custom location (does not modify Tomcat installation)

11. **Save** the configuration by closing this overview tab (click on the "x" of this tab) and confirming the changes if prompted by a dialog.

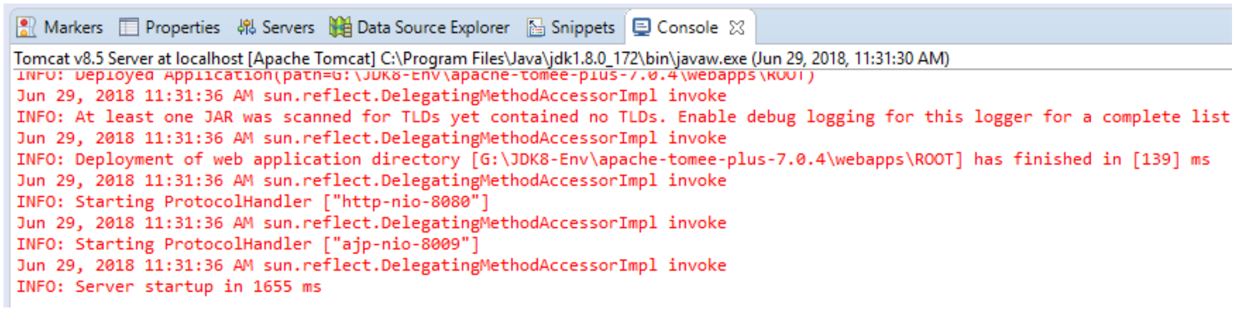
We recommend running TomEE in this configuration for a couple of reasons. First, this version of Eclipse seems to do a better job of coordinating, publishing, and adding/removing projects when the server is set up in this fashion. Secondarily, publishing the projects out to the TomEE installation makes the operation much more explicit and provides an opportunity to see the published applications in the file structure under the TomEE installation. Third, any configuration changes you make via this facility (e.g. port numbers, etc.) will remain effective even when TomEE is run stand-alone (i.e., without control from Eclipse).

12. Right-click on the server entry and select **Start**.



The server should start, with status messages appearing in the **Console** to reflect a successful startup.

13. During startup of the server monitor, watch the console for any errors that might occur during startup of the web server.



Tomcat v8.5 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (Jun 29, 2018, 11:31:30 AM)

```
INFO: Deployed Application(path=G:\JDK8-Env\apache-tomee-plus-7.0.4\webapps\ROOT)
Jun 29, 2018 11:31:36 AM sun.reflect.DelegatingMethodAccessorImpl invoke
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete list
Jun 29, 2018 11:31:36 AM sun.reflect.DelegatingMethodAccessorImpl invoke
INFO: Deployment of web application directory [G:\JDK8-Env\apache-tomee-plus-7.0.4\webapps\ROOT] has finished in [139] ms
Jun 29, 2018 11:31:36 AM sun.reflect.DelegatingMethodAccessorImpl invoke
INFO: Starting ProtocolHandler ["http-nio-8080"]
Jun 29, 2018 11:31:36 AM sun.reflect.DelegatingMethodAccessorImpl invoke
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
Jun 29, 2018 11:31:36 AM sun.reflect.DelegatingMethodAccessorImpl invoke
INFO: Server startup in 1655 ms
```

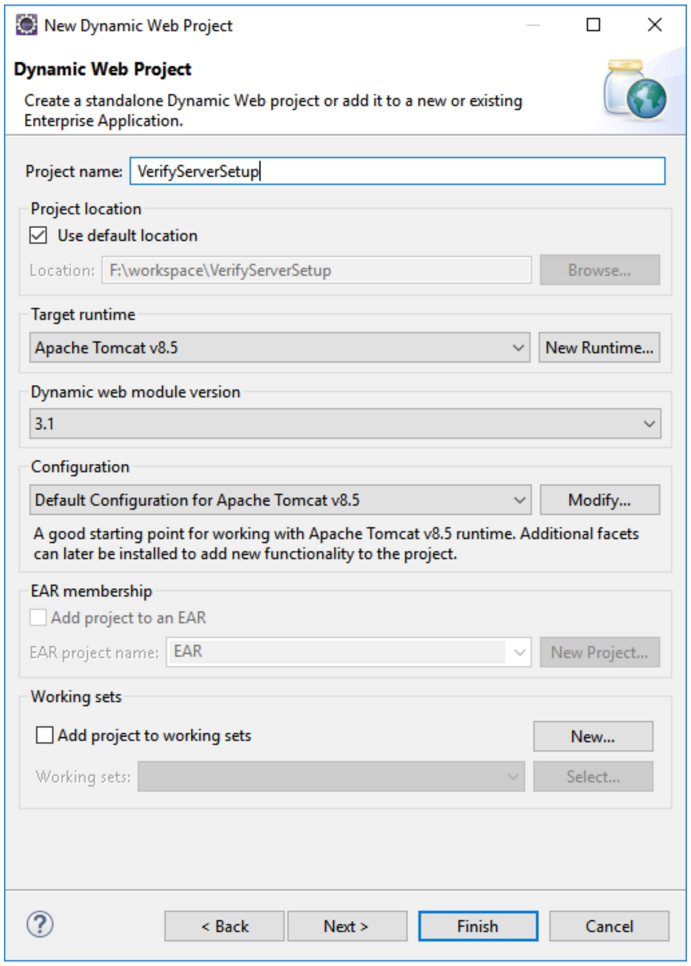
You might want to check this console regularly while deploying and running applications. Any errors that might occur during the deployment process will be displayed in the console.

14. To stop the server, you can click on the red square in the console or return to the ‘Servers’ view and right-click on the server entry and select **Stop**. The server status should change to **Stopped**.

Creating and Running a Web Project

1. Press **Ctrl+N** to open the **New** dialog. Select **Web -> Dynamic Web Project** and click **Next**.
2. Enter a project name, for example, of **VerifyServerSetup**. Your project name will vary based on the lab you are implementing. Check the lab instructions for the actual name.

   Ensure that the Target Runtime pulldown has Tomcat 8.5 selected.
3. Continuing from the **New Dynamic Web Project** dialog, ensure that **Configuration** is set to the default and **EAR membership** is **unchecked**. Click **Finish**.


New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: VerifyServerSetup

Project location

☑ Use default location

Location: F:\workspace\VerifyServerSetup

Target runtime: Apache Tomcat v8.5

Dynamic web module version: 3.1

Configuration: Default Configuration for Apache Tomcat v8.5

A good starting point for working with Apache Tomcat v8.5 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

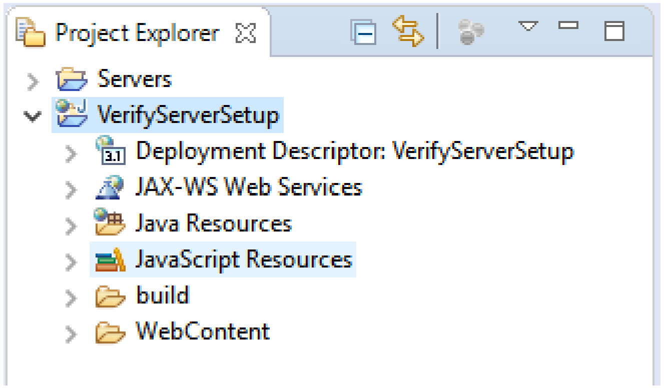
EAR project name: EAR

Working sets

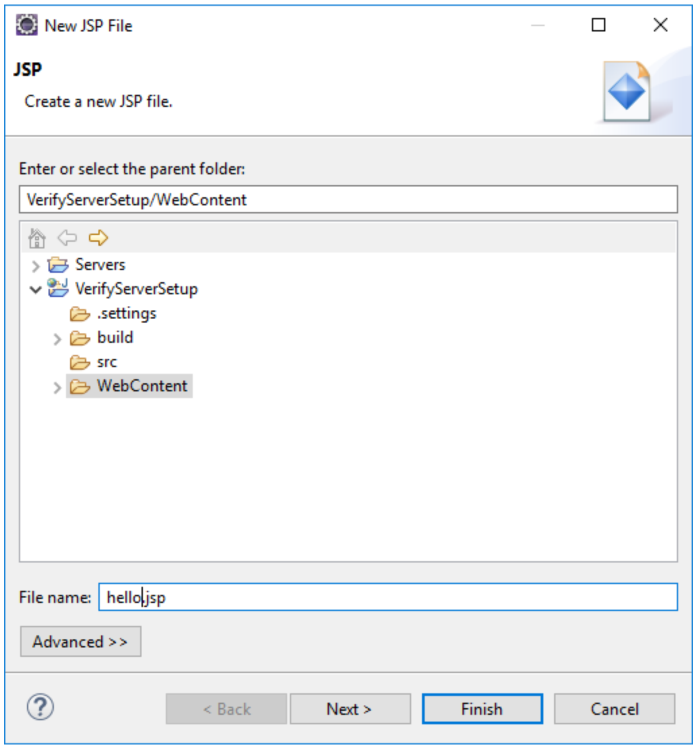
☐ Add project to working sets

Working sets:

4. If an **Open Associated Perspective** dialog box appears, accept the perspective change by clicking on **Yes**.
5. Your project will be listed in the **Project Explorer**.



6. Next, we'll create a simple **JSP** file to test the server configuration. In the **Project Explorer**, select the "node" of the newly created project and then press **Ctrl+N**. Select **Web -> JSP File**.
7. Select the parent folder **VerifyServerSetup/WebContent**. Enter a filename, such as `hello.jsp`. Click **Finish**.


New JSP File

JSP

Create a new JSP file.

Enter or select the parent folder: VerifyServerSetup/WebContent

File name: hello.jsp

8. When the `hello.jsp` file opens, enter some text in the **<TITLE>** and **<BODY>** elements. You can enter a simple string, such as Hello.

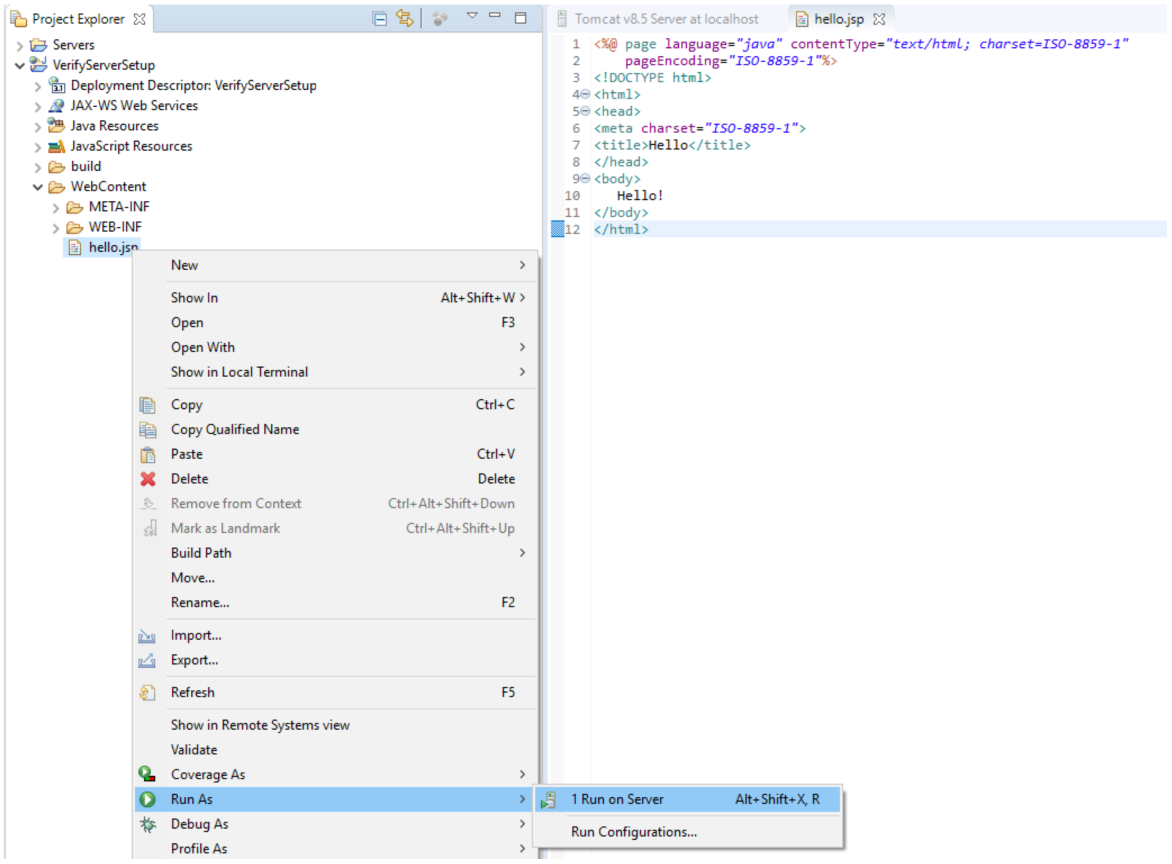
```
<HTML>
        <HEAD>
                <TITLE>Hello</TITLE>
        </HEAD>
        <BODY>
                Hello!
        </BODY>
</HTML>
```

9. **Save** the file.

In order to check the above file, a number of things have to happen:

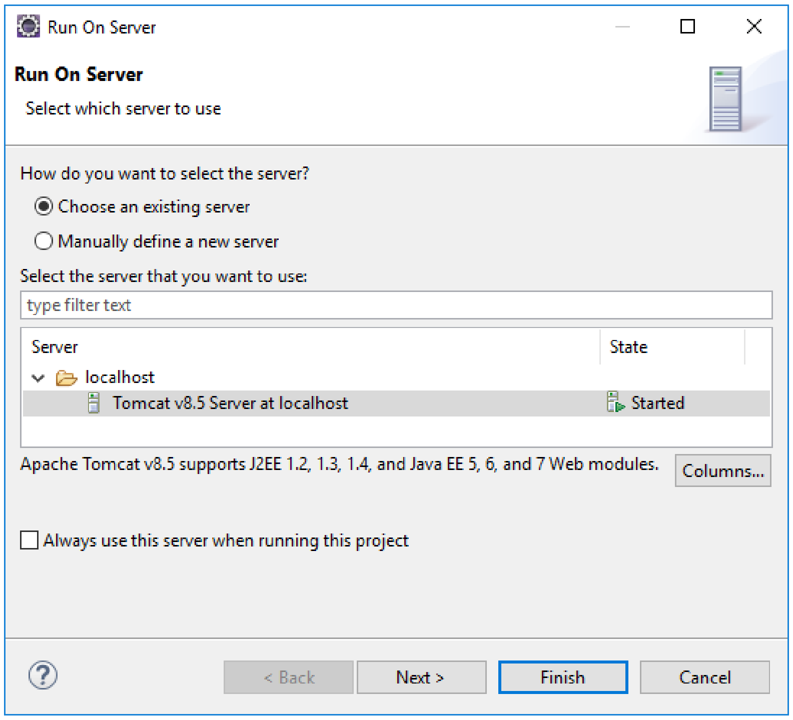
- The web server (application server) must be started.
- The application must be deployed to the web server.
- A web client (normally, a browser) must be started and it must send a request to the proper URL to cause the page to be executed and displayed.

10. All three of those tasks will be executed by selecting one popup menu item. Right-click on `hello.jsp` (you may have to open the **WebContent** folder to see it), and select **Run As -> Run On Server.**



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Hello</title>
</head>
<body>
   Hello!
</body>
</html>
```

11. The **Run On Server** window will open. If you have already defined a server for this project (essentially, a "deployment profile"), select **Choose an existing server**, make sure that the server you defined is selected.



Run On Server

Select which server to use

How do you want to select the server?

● Choose an existing server

○ Manually define a new server

Select the server that you want to use:

type filter text

Server	State
localhost	
Tomcat v8.5 Server at localhost	Started

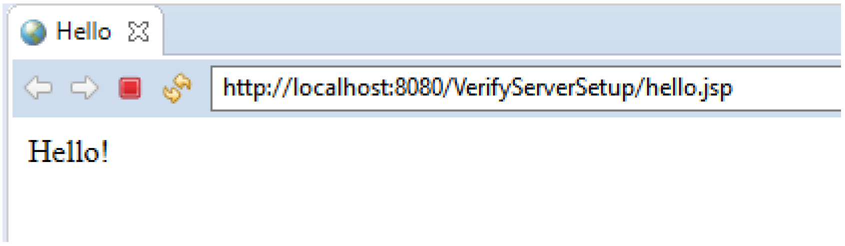
Apache Tomcat v8.5 supports J2EE 1.2, 1.3, 1.4, and Java EE 5, 6, and 7 Web modules.

☐ Always use this server when running this project

12. Click **Next** to check that the project is in the **Configured Projects** list. If it is not, then select it from the **Available Projects** list to the left and click **Add** to move it to the right.

13. Click **Finish**.

You will see build and deployment output quickly appearing in the **Console** view. The server is started and the application is deployed and started. When it completes, a web browser window will open (by default, the built-in Eclipse browser will create a new tab in the IDE's file-editor area) and display the string **Hello!.** It may take some time to deploy the application and start up the TomEE instance if it is not already running. Initially, the web browser may show an error message because the application is not yet deployed. After a few seconds, refresh the browser and the deployed application should be accessible.



Creating Another Web Project

14. On the workbench, select: **File ->New -> Dynamic Web Project**.
15. Enter **Tutorial** as project name and click **Finish**.
    The project name will be specified for each exercise (use the project name ' **Tutorial** ' for this exercise).
16. **Always target the appropriate runtime**.

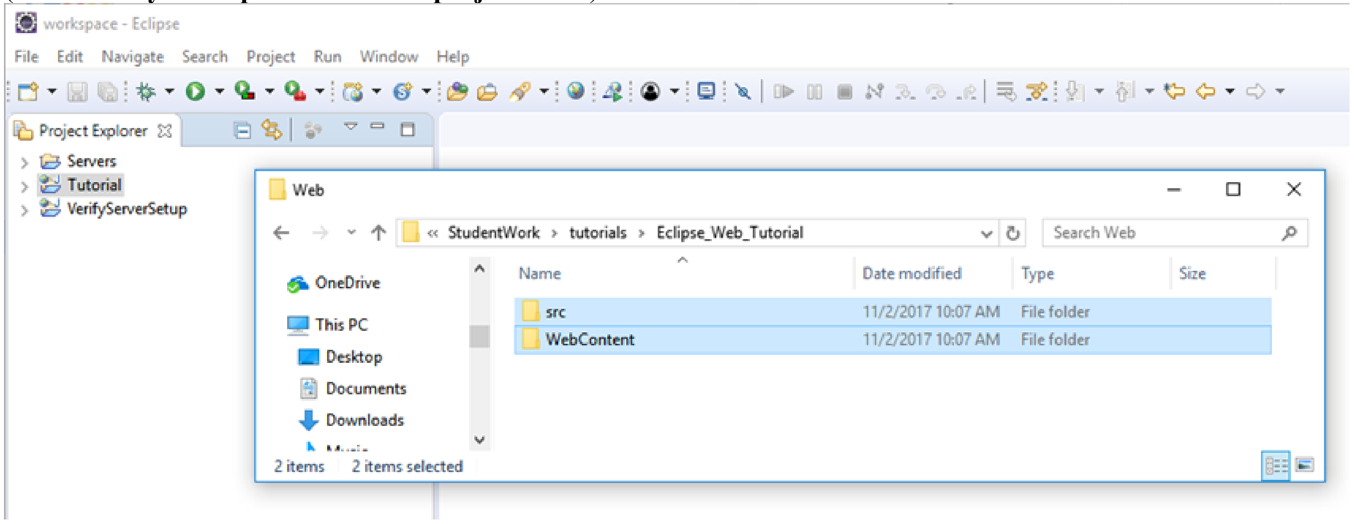
Note: By default, Eclipse uses the project name as the context root for the associated web application. In order to change this default, select **Next** twice and change the context root entry at the location prior to selecting **Finish**. To change the context root at a later time, **right-click** on the web project entry in the **Project Explorer**, select **Properties**, select **Web Project Settings**, and change the context root entry at that location.

Importing the Web Project

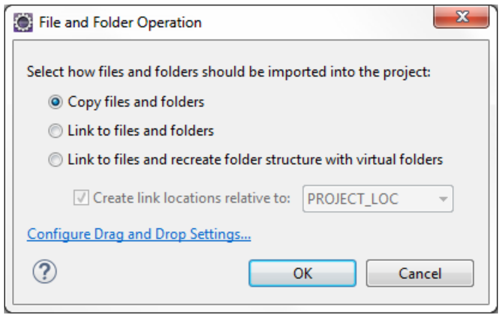
17. Using the Windows explorer, browse to the location of the exercise (for this tutorial**: ~Studentwork/tutorials/Eclipse_Web_Tutorial**)

You will notice that the content of the **Eclipse_Web_Tutorial** folder contains the same subfolders as the **Project** folder you just created in Eclipse.

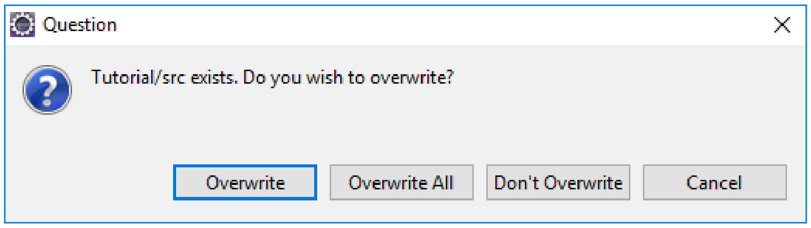
18. **Select all files and subfolders** and drag the content to the **Project** in which you would like to import the code. (**Make sure you drop the content at project level!**)



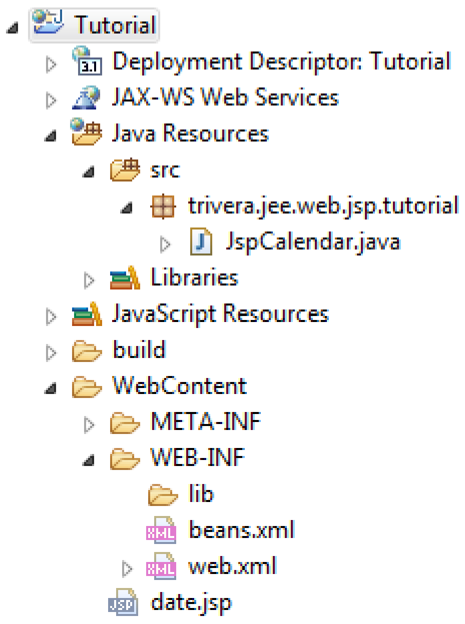
19. A **File and Folder Operation** pop-up will be displayed. Make sure the option **Copy files and folders** is selected and click **OK**.


File and Folder Operation
Select how files and folders should be imported into the project:
Copy files and folders
Link to files and folders
Link to files and recreate folder structure with virtual folders
Create link locations relative to:
PROJECT_LOC
Configure Drag and Drop Settings...
OK
Cancel

20. Eclipse will warn you to indicate that the project already has a **src** folders. Click **Overwrite All**.



21. After the import, your Tutorial project should look like this:

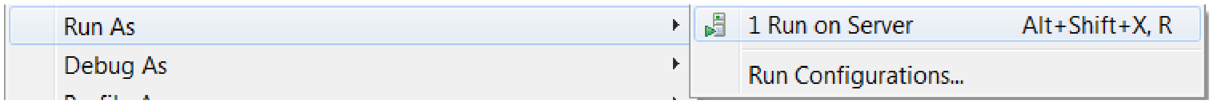


```
Tutorial
  Deployment Descriptor: Tutorial
  JAX-WS Web Services
  Java Resources
    src
      trivera.jee.web.jsp.tutorial
        JspCalendar.java
    Libraries
  JavaScript Resources
  build
  WebContent
    META-INF
    WEB-INF
      lib
      beans.xml
      web.xml
    date.jsp
```

Deploy your Application

22. In the **Project Explorer**, **right-click** on a web resource such as `date.jsp`.

23. Click on **Run As -> Run on Server**.



24. Select **Choose an existing server** and **Tomcat 8.5 Server @ localhost**.

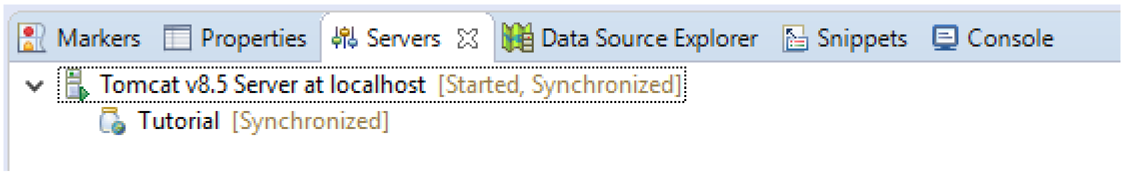
25. Click **Finish**. It may take some time to deploy the application and start up the TomEE instance if it is not already running. Initially, the web browser may show an error message because the application is not yet deployed. After a few seconds, refresh the browser and the deployed application should be accessible.

Examine Results

26. The application will be deployed on the server. If the deployment fails, the error will be displayed in the deployment panel.
27. Once the application has been deployed, Eclipse will start a browser, showing the welcome screen of your web application. For this application, a simple display of the date and time should appear similar to this:

```
• Day of month: is 24
• Year: is 2019
• Month: is May
• Time: is 11:38:26
• Date: is 5/24/2019
• Day: is Friday
• Day Of Year: is 144
• Week Of Year: is 21
• era: is 1
• DST Offset: is 1
• Zone Offset: is 1
```

28. In addition, you can access your web application by starting your default web browser and point to: **http://localhost:8080/Tutorial/date.jsp**
29. If the server was already running when you deployed a new web application, the server may need to be restarted. If so, here is how:
30. On the **Servers** panel, **right-click** on the server entry and choose **Restart**.
31. You must wait until the server status on the **Servers** pane indicates **Started** before you can test your application.



Tomcat v8.5 Server at localhost [Started, Synchronized]
Tutorial [Synchronized]

On the **Console** pane, the server will also indicate when it's capable of handling requests.

If you are successful in importing and deploying this web application, you will see today's date and time displayed in the browser.

Redeploy Your Application

When you make changes to your application after it has been deployed, your application does not always have to be deployed again. The server may pick up updates automatically when the application is rebuilt. If the server does not seem to have picked up the changes, here is what to do:

32. Make sure all the edited files have been saved.
33. In the **Server** view, right click on the targeted server entry and select **Publish** to cause the application to be redeployed.
34. If that does not cause your changes to be deployed, rather than select **Publish**, select **Add and Remove Projects**, remove the project, and select **Finish** to get TomEE to undeploy the application. Once that has completed, follow the same steps to add the project to the server and deploy it.

TUTORIAL: WORKING WITH THE HSQL DATABASE

Preface

For this course, we will running the HSQL database as a server. This open source server must be started prior to connecting to it. The server must be properly shutdown in order to persist changes across database sessions.

🛠 Starting the HSQL Database

☐ 1. This HSQL database is run as a server by opening a DOS command window on the **StudentWork/HighViewDB** folder and running **start_highview**. Leave the DOS window open as that is the runtime output for the database server (status and error messages will appear in this window).

🛠 Stopping the HSQL Database

☐ 1. Stop the HSQL server by opening a DOS command window on the **StudentWork/HighViewDB** folder and running **stop_highview**.

NOTE: If you would like to shutdown the database without persisting changes, close the DOS window that you originally ran the database from. This will shutdown the database in an abnormal fashion and nothing will get persisted.

Also note that there is a clean version of the database that can be used to "reset" the database back to its original state. The database is constructed using the **highview.script** file that is in the **StudentWork/HighViewDB/data** folder. To reset it, overwrite scripting file with the version of **highview.script** from the **StudentWork/HighViewDB/data/reset** folder and restart the database.

Exercise 1. CASE STUDY SETUP AND REVIEW

Overview	
You will examine a description and implementation of an application that we will be using during much of the remainder of this course. You will then import the application resources into your IDE, deploy it to your server, and explore how the application works.	
Objective	Understand the case study for this course as well as how to work with and run it.
Builds on Previous Labs	None
Approximate time	40 minutes

Task List

Overview:

Step 1: Review background of application
Step 2: Import and deploy the InsecureWeb application
Step 3: Examine implementation architecture

Note: Take your time with this lab, examining the components, how they work, and what they do. This will be very beneficial for subsequent discussions and activities.

🛠 Step 1: Review background of application

☐ 1. Spend a few minutes and read through application description

RHH Insurance has an existing automobile policy system that is supported by a set of databases. The system currently receives requests for copies of auto policies in a variety of formats and forms that must be handled on an individual basis. These policy requests come from independent agents. Automobile policy information is currently printed and sent out as hardcopy for each request. Policy information is also available to specific internal organizations through a highly proprietary client/server system. The decision has been made to migrate to a web enabled application.

Functional Requirements:

- Agents should be able to log on to the system using a user id and password.
- An agent's login session should last as long as the agent is actively requesting policies. In other words, once an agent has logged in, he or she should not have to log in again to request another policy unless the login session has timed out.
- There should be a time out capability that will end a session after a predetermined period of inactive time.
- There should be the ability to request a specific policy by policy number.
- There should be a consistent error-handling mechanism to provide feedback to the agent when there are problems.

- Agents and technical support personnel have indicated a desire for a HTTP diagnostics capability. This capability would not require a login and would return an HTML rendering of the information contained in the HTTP request sent to the server.
- Agents should be able to update their passwords as well as their information profile and email.
- Agents should be able to query for the email addresses and information profiles for other agents.

🛠 Step 2: Import and deploy the InsecureWeb application

☐ 1. Create a dynamic web project called **"InsecureWeb"**

☐ 2. Import the InsecureWeb application that can be found at **"StudentWork/Labs/CaseStudy"**.

☐ 3. Deploy the web application to your sever

☐ 4. Check the application using the correct context root (**http://localhost:8080/InsecureWeb**).

🛠 Step 3: Examine implementation architecture

☐ 1. We are now going to fast-forward from the functional description and assume that a design has been implemented for the insurance policy viewing application.

☐ 2. The entire implementation of this application is contained in a single web application. The entry point for the web application is the index.html page.

The index.html page has a series of links to various parts of the application. These links are setup and organized to meet the needs of this course rather than a logical grouping for a real application. We will be using all of these links and their associated functionality at various times in this course.

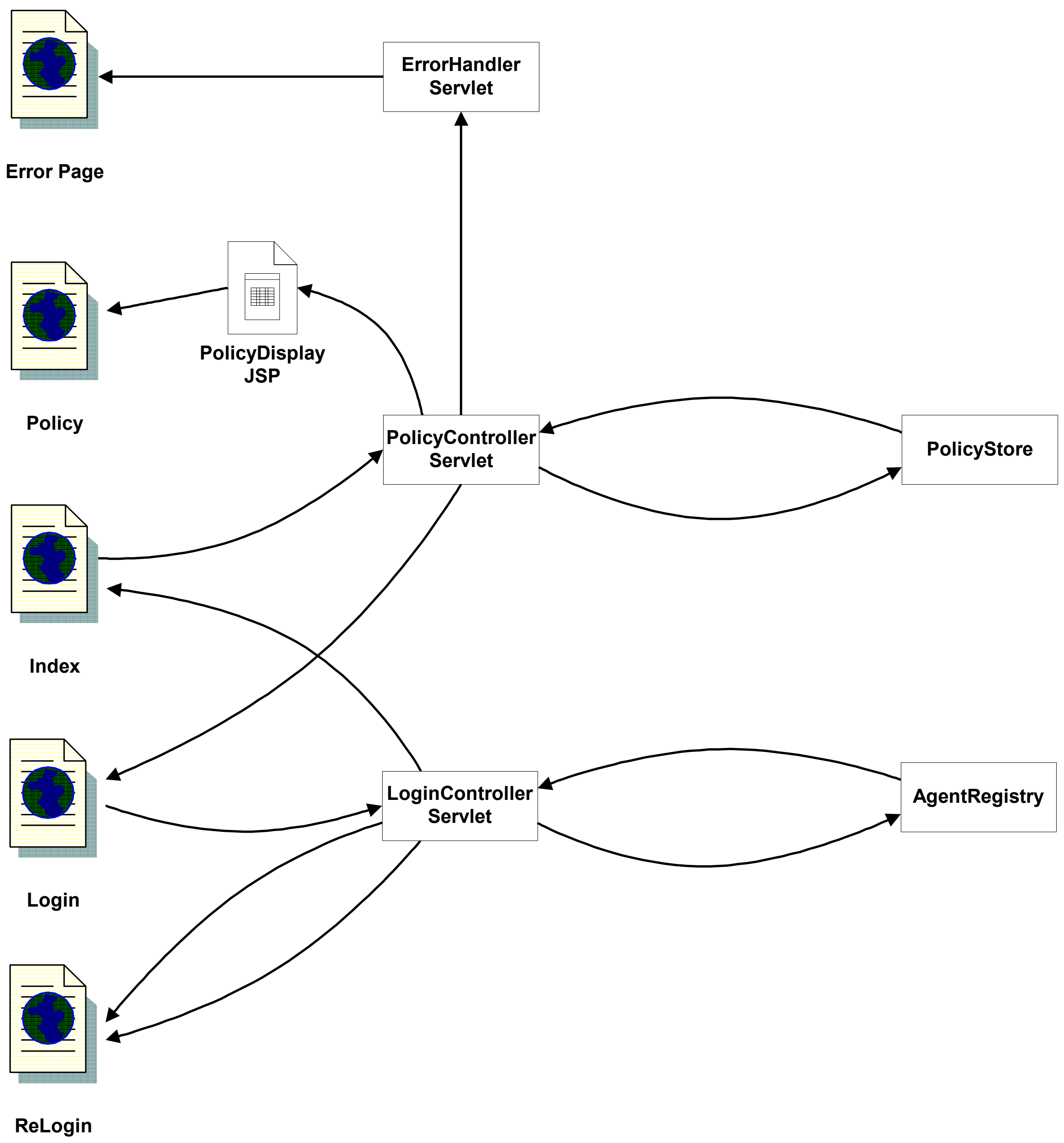
The links and functionality do meet the functional requirements that you reviewed in the first step in this exercise. All functionality is in place and working. Please examine the functionality and threads of processing for each set of links, verifying to your satisfaction that the application performs as advertised:

- Updating and querying for user information uses the contents of the HSQL database that you started a few minutes ago. You can examine the **highview.script** file that is in the **StudentWork/HighViewDB/data** folder to see what users and information is already in the database. Some existing users include test, dan, carlos, tresa, etc. You can only update existing users via this functionality. This does not support creating new users.
- The login capability is used in conjunction with other parts of the application and also runs against the HSQL database. Existing username/passwords include **test/test**, **dan/idaho**, and **carlos/hiking**. The update capability allows you to update the password for an existing user.
- Updating and querying for emails uses the contents of the HSQL database as well. Some existing users include test, dan, carlos, tresa, etc. You can only update existing users via this functionality.
- In addition, there is a reasonably complex web application for supporting a query and display of insurance policies. This search functionality requires a login and the use of sessions to capture the fact that a user has successfully logged in. The development team has provided a high level flow of control diagram that is included on the next page.

The datastore for the policies is a set of serialized objects that have been written out to a data file and are read in during initialization.

For search purposes, the valid policy numbers are 1, 2, 3, and 4.

- Finally, there is a button to submit to an existing DiagnoseHTTP servlet, which, in turn, generates an HTTP Diagnostics HTML page.



Exercise 2. CASE STUDY ASSET ANALYSIS

Overview	
Working in a team, you will analyze the case study, identifying what assets this particular application has.	
Objective	Develop an initial list of targets for attacks.
Builds on Previous Labs	`InsecureWeb`
Approximate time	60 minutes

Task List

Overview:

Step 1: Organize team and roles for team
Step 2: Develop list of assets
Step 3: Review list of assets with class

🛠 Step 1: Select team and roles for team

Purpose: Set a context for this exercise and organize the team you will be working with

☐ 1. Students should split into groups of 3-5 (depending on class size).

☐ 2. Once you have gotten your team together, select **one** person to be the team leader/application architect.

i. The primary reason for selecting one person as the team leader/application architect is to push the process and make decisions. This is an artificial application - questions will come up for which there is no obvious answer from the application description that is provided. The team leader/application architect will make those decisions (fairly quickly).

ii. A second person will need to be designated as the scribe and time keeper. In a real modeling process, this is often the developer (especially during an initial modeling meeting where things are fairly high level).

iii. One of the jobs of the scribe is to capture and maintain a list of results that are identified during the early stages of the modeling process.

🛠 Step 2: Develop list of assets

☐ 1. Using the case study as the basis for these discussions, identify as many assets as you can. Examine both the static (architecture and components) as well as the dynamic (functionality) aspects of the application. Students should spend 30 minutes on this activity.

☐ 2. When complete, each team should present their findings, with the instructor documenting their results, preferably on post-it type flip-chart pages that can remain up on the walls for the remainder of class (we will be updating these later).

Exercise 3. DEFENDING TRUST BOUNDARIES

Overview	
Using the InsecureWeb application, you will identify a set of trust boundary locations and the information that is crossing those locations. You will then develop a specification for each of the trust boundaries. Finally, you will implement an initial set of defenses for each of those trust boundaries.	
Objective	Gain experience defending against unvalidated input
Builds on Previous Labs	`InsecureWeb Project`
Targeted Files	UserQueryController.java UserUpdateController.java
Time to Complete	90 minutes

Task List

Overview of steps:

Step 1: Identify the a set of trust boundaries
Step 2: Develop a specification for each of the web-based trust boundaries
Step 3: Implement a defense for each targeted trust boundary
Step 4: Explore and use HTTP tools
Step 5: Review regular expression cheat sheet
Step 6: Install and explore the RegEx Util
Step 7: Use regular expressions in defending your trust boundaries
Step 8: Challenge Step - Refactor regular expressions

🛠 Step 1: Identify the set of trust boundaries

☐ 1. We are going to focus on the functionality and information associated with the **Query for User Information** and **Update User Information** links in the case study.

- Identify the trust boundaries and the entry/exit points for data through those boundaries for these two threads of execution. Trust boundaries are critical items to identify because they are principal attack targets.

Questions that can assist in determining entry points:

- What data acquisition mechanisms are going to be used?
- What data needs to be acquired and how does it enter the system?
- What inner data entry points can be identified at this point? (These are data entry points between internal components that can help clarify where trust boundaries need to be.)

Questions to ask in helping identify and analyze exit points include:

- Where does data exit the system (back to users or to external systems)?
- Does data from an entry point get passed through the exit point?

🛠 Step 2: Develop a specification for each of the web-based trust boundaries

☐ 1. Create a new folder in your **InsecureWeb** project called **Design**.

☐ 2. Import the file **Data Spec.doc** into the **Design** folder. This file can be found at **~/StudentWork/Labs/TrustBoundaries**

☐ 3. Identify the web-based trust boundaries from the previous step

☐ 4. For each of these trust boundaries, define a robust, positive specification of expected values. Use the **Data Spec.doc** to capture your design specs. In some cases, you may not know exactly what is allowed for the application. Decide what would make realistic sense and use that as a basis for your specification.

🛠 Step 3: Implement a defense for each targeted trust boundary

☐ 1. Using the codebase that you previously examined and ran, implement an initial defense for each trust boundary. Note: The servlet classes that handle the user information functionality include a utility called **genBadIdPage** that can be used to handle error conditions.

☐ 2. Test your approach for all specified criteria.

🛠 Step 4: Explore and use HTTP tools

☐ 1. We will be using some freely available tools to examine and manipulate content being submitted to web applications. The first of these tools works with either Internet Explorer or Firefox. The second is an extension to Firefox and only works with that browser.

☐ 2. Fiddler is a powerful tool that can be used with either Internet Explorer or Firefox, although it is more tightly integrated with Internet Explorer.

This is a tool with considerable functionality as well as documentation and a user base that can be found on the web. We will only be scratching the surface of what the tool can do.

Note: Fiddler should already be installed on your computer. If you do not see the **Fiddler** in the Windows Start Menu then it has not been installed. Let your instructor know about the situtation.

To run Fiddler: Start the tool from the Windows Start Menu. Once it is started, all internet traffic from any browser will be routed through the tool.

☐ 3. Using your browser, navigate to a web site and note how Fiddler is capturing the requests and responses. The **Inspectors** and **Statistics** tabs (on the right side of the interface) provide views of what is going on. Check them out.

☐ 4. Note that you can select a request entry on the left side (right-click on it) and replay the request. This will submit the request from Fiddler. The response will be captured in the **Inspectors** tab. If you select the **WebView** entry, you can see what that specific result looks like rendered in a browser.

☐ 5. If you have a local server running, you will need to change the URL in order to force traffic through the proxy. There are two ways to do this. First you can add a period to the end of localhost: **http://localhost.:8080**. The alternative is to put your computer's network name into the URL in place of localhost. Once you start using one of these forms, you will be able to monitor local traffic as well. Make sure that you can access your local server via Fiddler.

☐ 6. Note that there is a tab called **Composer**. Select that tab. As you can see, you can construct your own HTTP Request (Get or Post) and submit it. The results of that submission can be examined in the **Inspectors** tab.

☐ 7. You can drag a Request entry from the left side into the **Composer** and use that as a prototype to submit (attack) a web application. You can edit the values in the request and submit the modified request as well. Fiddler has additional capabilities for submitting entire files of requests as well as scripting available.

🛠 Step 5: Review regular expression cheat sheet

☐ 1. You will find a cheat sheet for regular expressions in the **StudentWork/Labs/RegularExpressions** folder in the files called J**avaRegularExpressionsCheatSheet.pdf**. This is a regular expression cheat sheet that summarizes the basic syntax for constructing regular expressions.

🛠 Step 6: Install and explore the RegEx Util

☐ 1. One of the tools that you may use for working with regular expressions is a plugin called the **RegEx Util**. This can be found in the **StudentWork/Tools** folder as the **com.ess.regexutil_1.2.4.jar** file.

☐ 2. To install the plugin:

- Exit Eclipse if it is currently running.
- Navigate to the **~/StudentWork/Tools** folder and locate the **com.ess.regexutil_1.2.4.jar** file.
- Copy the JAR file into **plugins** folder under the IDE installation directory (for example, **C:/eclipse/plugins**).
- To verify that the plugin is in the correct locations, navigate to the plugins folder under the IDE installation directory and you should see the following file: **com.ess.regexutil_1.2.4.jar**
- Restart Eclipse

☐ 3. The **RegEx Util** view is accessed via **Window -> Show View -> Other -> RegEx Util**, which opens up a RegEx view. One advantage of this particular tool is that it has a simple, intuitive interface that is easy to use in developing and testing regular expressions. It also uses the Java regular expressions implementation. Here are some features of this tool:

  - o Matches are colorized, for an easy visual clue
  - o Support for pattern flags (e.g. Pattern.DOTALL)
  - o The tool evaluates your regular expression while you are typing it, gives feedback on possible errors and shows any matches automatically

☐ 4. Start with something extremely simple such as a Regular Expression of "test". Now type the word "test" into the window immediately below the Regular Expression window. This time you should get a match when you evaluate the expression.

☐ 5. Start with a simple expression such as ^(19|20)[\d]{2,2}$, which could be used to ensure that the targeted text corresponds to a year between 1900 and 2099. Enter the expression into the text box labeled **Regular Expression**. Try various text strings in the text box, watching the output of evaluating the expression in the result box. When you enter a string that matches the test expression, the tool highlights the matching string.

🛠 Step 7: Use regular expressions in defending your trust boundaries

☐ 1. Using Java regular expression capabilities and the techniques described in the associated lecture material, develop a set of regular expressions for defending the trust boundaries associated with the with the **Query for User Information** and **Update User Information** links in the case study. These are, of course, the trust boundaries that you developed initial defenses for in the last lab.

☐ 2. Test each of the applied expressions.

☐ 3. Given that regular expressions are relatively expensive to apply (from a computational perspective), does it make sense to only rely on them for defending your trust boundaries?

🛠 Step 8: Challenge Step - Refactor regular expressions

☐ 1. Once you get the basic regexp defenses in place and tested, you are welcome to take the implementation one step further. Often times, the regular expressions themselves are extracted from the code and placed into initialization parameters in the web.xml file. The advantage of this is that the defenses can be adjusted without having to change the code and recompile it.

☐ 2. Since regular expressions can contain XML-sensitive characters, the strings themselves must be encapsulated in CDATA sections in order to parse properly.

☐ 3. Extract your regular expressions and place them as initialization parameters in the web.xml file.

NOTE: The solutions for each of the programming labs are contained in the **~/StudentWork/Solutions** folder. Each solution only has those part of the InsecureWeb application that were affected by the associated lab. If you would like to build up a solution set for the course, you can create a Dynamic Web Project called **Solution** and in then import the code that you originally imported into the InsecureWeb project.

As you complete each lab and are ready to examine the solution, import the solution code into the solution project. This way, you call look at and run each incremental solution.

Exercise 4. DEFENDING AGAINST SQL INJECTION

Overview	
During this exercise, you will explore SQL Injection. You will try various attacks against the application. You will then implement defenses against these attacks.	
Objective	Gain experience recognizing and defending against SQL Injection.
Builds on Previous Labs	`InsecureWeb Project`
Targeted Files	EmailQueryController.java web.xml
Time to Complete	60 minutes

Task List

Overview of steps:

Step 1: Explore the SQL injection attacks
Step 2: Design and implement an effective SQL Injection defense
Step 3: Test your defenses

🛠 Step 1: Explore SQL injection attacks

☐ 1. For this lab, we will be focusing on the email query functionality in our case study. Examine the functionality associated with the **Query Email** functionality. As you follow the thread of execution, you will see that the a simple SQL query statement is executed against the HSQL database. Test the functionality, using the **Query for Email** link.

☐ 2. For this lab, we are interested in whether the email query and associated functionality is vulnerable to SQL injection attacks. Start out by trying the example from the course notes **‘ OR 1=1 --**

☐ 3. Now try the following attacks (these can be found in electronic form in the **SQLInjectionExamples.txt** file in the **StudentWork/Labs/SQLInjection** folder)

‘ OR 1=1 UNION ALL SELECT USERNAME FROM MEMBER UNION ALL SELECT PASSWORD FROM MEMBER --

‘; INSERT INTO MEMBER VALUES('12345678','BigDummy','sleeping','home','Cloud9','10019','U.S.A.','1010-695-3405','AAE','BigDummy@DummyTime.COM','A',NULL); --

'; SHUTDOWN; --

Note that you must not place line returns in these values.

Also note that if you try and run the INSERT command multiple times, it will throw an exception. You will need to adjust the contents slightly to get a unique insertion.

🛠 Step 2: Design and implement an effective SQL Injection defense

☐ 1. Shift the current implementation for database interactions to using PreparedStatements. (For those that need some hints on a PreparedStatement, there is an example in the **StudentWork/Labs/SQLInjection** folder.)

☐ 2. Develop rigorous, positive specifications for the trust boundary.

☐ 3. Implement defenses that are strong enough to effectively defend against active code and malicious scripts being injected via this trust boundary.

🛠 Step 3: Test your defenses

Purpose: Verify your measures are effective.

☐ 1. Test your trust boundary defenses to ensure that they are working effectively. Ensure that the trust boundary defenses prevent the malicious input from being used in an SQL query. Once you have validated that layer of defenses, disable them.

☐ 2. Test that the use of a PreparedStatement also prevents the SQL injection attack. Ensure that, despite that fact that the malicious code is injected into the PreparedStatement, it does not cause problematic behavior. Once you are satisfied with that, reenable your trust boundary defenses.

☐ 3. Determine if your injection defenses that you have built are vulnerable to XSS attacks. Deactivate your boundary defenses and attack the PreparedStatement. Then reactivate your defenses and verify that they are working.

Challenge: Using examples from the lecture, modify your PreparedStatement so that it is susceptible to an injection attack. Verify that you can successfully attack a PreparedStatement.

Bonus Question: Would a Web Service be vulnerable to an SQL Injection attack?

A successful SQL Injection attack requires two factors to be in place:

Data received from an untrusted source that is inserted directly into a SQL statement

The SQL Statement is run in the context of a user with sufficient privileges to execute the attack.

Let's look at an example:

The following SOAP message is received:

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header></SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <BookLookup:searchByIBSN
             xmlns:BookLookup="https://www.books.com/Lookup">
      <BookLookup:IBSN>0072224711<BookLookup:IBSN>
    </BookLookup:searchByIBSN>
  </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

This message is processed by the following VB.NET code, which inserts the content of the IBSN element into a SQL statement.

```
Set myRecordset = myConnection.execute("SELECT * FROM myBooksTable
WHERE
IBSN ='" & IBSN_Element_Text & "'")
```

In the case of the SOAP message above, this becomes:

```
SELECT * FROM myBooksTable WHERE IBSN = '0072224711'
```

Let's say the following SOAP message is received:

```
<SOAP-ENV:Envelope
     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header></SOAP-ENV:Header>
  <SOAP-ENV:Body
    <BookLookup:searchByIBSN
             xmlns:BookLookup="https://www.books.com/Lookup">
      <BookLookup:IBSN>
       0072224711'; exec master..xp_cmdshell 'net user Joe pass /ADD';
--
      <BookLookup:IBSN>
    </BookLookup:searchByIBSN>
  </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

In this case, the SQL statement will read:

```
SELECT * FROM myBooksTable WHERE IBSN = '0072224711'; exec
master..xp_cmdshell 'net user Joe pass /ADD'; --
```

The code after the SELECT statement attempts to create a user called "Joe" with password "pass". An attacker could then attempt to use this new user account to gain access to the target machine.

Exercise 5. DEFENDING AUTHENTICATION

Overview	
During this exercise, you will design and implement defenses for several aspects of the authentication portion of this application. You will: • Develop a username and password policy • Defend the login functionality using that policy • Setup a basic rate-limiting defense for the login functionality • Enforce the password policy in the password update functionality	
Objective	Understand various aspects of protecting authentication
Builds on Previous Labs	`InsecureWeb Project`
Targeted Files	AgentRegistry.java LoginController.java PwdUpdateController.java web.xml
Time to Complete	60 minutes

Task List

Overview of Steps:

Step 1: Review InsecureWeb password management
Step 2: Develop a username and password policy
Step 3: Defend the login functionality
Step 4: Design and implement a basic rate limiting defense
Step 5: Incorporate the username/password policy into update process
Step 6: Challenge - Integrate hashing functionality into password generation
Step 7: Challenge - Integrate hashing functionality into authentication process

🛠 Step 1: Review InsecureWeb password management

☐ 1. Examine the following aspects of password management in the InsecureWeb application:

a. Mechanism for initially setting up (or modifying) a password

b. Authentication data repository

c. Mechanism for performing authentication checking

🛠 Step 2: Develop a username and password policy

☐ 1. Develop and write down a policy for username and passwords. Specify what you would expect for minimum and maximum limits as well as alphanumeric requirements. Capture your design the **Data Spec.doc** that you used previously.

☐ 2. Prior to continuing with this lab, use the Username and Password update functionality to ensure that you have at least one user that meets the specification.

🛠 Step 3: Defend the login functionality

☐ 1. One of the problems with the current login functionality is the differing error response between an invalid username as opposed to an invalid password. Redesign and implement this error response to transform it into a defensively correct response.

☐ 2. Using the username and password policies as your specification, implement initial trust boundary defenses for the login functionality. Test those defenses to ensure they are working properly. What would be the defensively correct error response to use for these defenses?

NOTE: You will need to ensure that there is at least one username and password in the database that fits your defense criteria. If required, you can edit the source script for the database that is contained in **StudentWork/HighViewDB/data**

🛠 Step 4: Design and implement a basic rate limiting defense

☐ 1. Design a basic rate limiting defense of the login functionality. For the moment, design this defense to lock out a particular user that has not logged in successfully three times. Ensure that the mechanism is both efficient and thread-safe.

☐ 2. Implement and test the defense.

☐ 3. Rate limiting defenses are always potential mechanisms for exploiting a denial of service attack. This is especially true if the username policy is weak or predictable.

- One way of mitigating this (to an extent) is to lock out the person for a limited amount of period of time. How would you extend your existing mechanism to facilitate this added capability?
- Another mitigation approach is to gradually increase the lock out period as the user fails more attempts. For example, you might allow a person another opportunity after 5 minutes then 10 minutes and so on.

☐ 4. Another potential denial of service vulnerability is in error response on the server side. Determine whether there are any error or status messages being written out. Any error or status messages associated with authentication should be examined for content to ensure that no sensitive information is being written out. In addition, the cost of writing those messages out and maintaining the associated message store can also be a target for denial of service. Correct any vulnerability that you encounter.

🛠 Step 5: Incorporate the username/password policy into update process

☐ 1. You need to build defenses for the trust boundaries associated with the **Update Username/Password** functionality. Those defenses should exclude invalid characters and should incorporate an implementation of the username and password policies.

☐ 2. Incorporate the username and password policies into the defenses that you need to build for the **Update Username/Password** functionality. Implement and test those defenses.

🛠 Step 6: Challenge - Integrate hashing functionality into password generation

☐ 1. Examine the hashing example provided in the **StudentWork/Labs/Authentication** folder. The code is liberally commented to explain how the hashing works.

☐ 2. Using the example as a basis, integrate hashing functionality into the password generation functionality.

☐ 3. You will need to add the external jar, **bcprov-jdk15-133.jar**, to the project. This JAR file can be found in **StudentWork/Labs/Authentication**

☐ 4. Verify that submitted passwords are being stored in a hashed form.

🛠 Step 7: Challenge - Integrate hashing functionality into authentication process

☐ 1. Using the example as a basis, integrate hashing functionality into the authentication functionality.

☐ 2. Verify that submitted passwords are correctly accepted and rejected.

☐ 3. Verify that using the stored version of the password does not authenticate.

☐ 4. If a user loses his or her password, how would you respond to a request for the original password?

Exercise 6. DEFENDING SENSITIVE DATA

Overview	
During this exercise, you will examine the case study to identify sensitive data. You will then propose some defenses for that data and, optionally, implement them	
Objective	Understand some of the complexities involved in handling sensitive data
Builds on Previous Labs	`InsecureWeb`
Time to Complete	60 minutes

Task List

Overview of Steps:

Step 1: Identify sensitive data that is being used in the Case Study
Step 2: Design defenses for each instance of sensitive data
Step 3: Challenge - Integrate encryption functionality
Step 4: Challenge - Integrate decryption functionality

🛠 Step 1: Identify sensitive data that is being used in the Case Study

☐ 1. Review the InsecureWeb application in its entirety and identify all data associated with the application that is sensitive. Your review should include the data that the application works with as well as any information that the application uses as part of its runtime operations.

🛠 Step 2: Design defenses for each instance of sensitive data

☐ 1. List actual or proposed defenses for each of instance of sensitive data that you have identified.

☐ 2. Once you have completed that list, your instructor will discuss them with the class. If you are so inclined, you can study the next, optional, challenge steps.

🛠 Step 3: Challenge - Integrate encryption functionality

☐ 1. Examine the encryption example provided in the **"StudentWork/Labs/Encryption"** folder. The code is liberally commented to explain how the encryption works.

☐ 2. Using the example as a basis, integrate encryption functionality into the your application to use in protecting one of the instances of sensitive data.

☐ 3. Verify that persisted data is being stored in an encrypted form.

🛠 Step 4: Challenge - Integrate decryption functionality

☐ 1. Examine the decryption example provided in the **"StudentWork/Labs/Encryption"** folder. The code is liberally commented to explain how the decryption works.

☐ 2. Using the example as a basis, integrate decryption functionality into your application to decrypt the sensitive data that you encrypted and persisted in the previous step.

☐ 3. Verify that the entire process works as you would expect. What vulnerability concerns remain relative to this defense?

Exercise 7. SAFE XML PROCESSING

Overview	
During this exercise, you examine various potential security concerns that are related to the basic processing of SOAP messages.	
Objective	Gain an appreciation of the vulnerabilities associated with web services and SOAP.
Builds on Previous Labs	`None`
Targeted Files	SAXTool.java
Time to Complete	40 minutes

Task List

Overview of Steps:

Step 1: Analyze each of the SOAP messages
Step 2: Design and implement defenses for XML parser

🛠 Step 1: Analyze each of the SOAP messages

☐ 1. The first step in processing any XML is going to be the parsing of the XML document. The parse may be validating or non-validating and maybe with a SAX parse or a DOM parse. There are many constructs within XML that cannot be excluded or prevented from being used (for example, there is no way to prevent a CDATA section from being included within the content). The primary options that you have to adjusting standard parsing behavior is:

- Use the parser options and features to prevent or preclude certain types of potentially unsafe operations (such as resolving external entity references)
- Use of the safe XML processing options introduced in Java 1.5 can help address some of these
- Some issues can be dealt with by performing a SAX parse and handling appropriate events in a specialized fashion.

☐ 2. Navigate to the **"StudentWork/Labs/SOAP"** folder and examine the contents.

☐ 3. The best way to observe the parser behaviors is to open each example in one or more browsers such as Internet Explorer or FireFox. A browser will attempt to perform a non-validating DOM parse on an XML document, processing any DTD constructs as well since entity references may need to be resolved during XML parsing. Once a DOM is constructed any referenced stylesheets will be processed and applied to the browser for rendering purposes. Examples include:

- **personal.xml**: A basic, successful parse of this small 2K XML will be completed and a expandable XML tree will be displayed in the browser.

- **menuProcess.xml**: XML document that successfully parses, but the referenced XSL stylesheet is not at the expected location and the transformer throws an exception and stops processing. This illustrates that externally referenced resources may be used to perform denial of service attacks.

- **personalBadDTDRef.xml**: XML document that fails to successfully parse...not even the DOM is constructed in this case. By default, an XML parser will dereference and attempt to parse any referenced DTD. Any incoming XML document (including SOAP messages) could be afflicted with this problem. Note that FireFox does not throw an error in this case, while Internet Explorer does.

- **personalBadDTD.xml**: XML document that takes a inordinate amount of time to successfully parse. By default, an XML parser will parse and process any internal subset and then resolve any entity references in the XML. Any incoming XML document (including SOAP messages) could be afflicted with this problem. Note that FireFox is considerably faster processing this example.

- **personalBig.xml**: XML document that brings in a large amount of additional content from another location (could be a URL at a remote site). Any incoming XML document (including SOAP messages) could be afflicted with this problem. Note that FireFox does not throw process the external reference, while Internet Explorer does.

🛠 Step 2: Design and implement defenses for XML parser

☐ 1. Create a new Java Project called **DefendingXML.** Import the resources at **~/StudentWork/Labs/SOAP/src** into the project's source code folder.

☐ **2.** This codebase is a tool normally used for teaching how to programmatically work with XML and XSLT. You can run it as a Java Application by selecting the `com.triveratech.training.xml.kit.XMLTrainingKit` and runnning it.

☐ **3.** There are several example XML files that were imported with the JAR file that you can parse with the tool. Try them out and observe the behavior of the parser. In particular, parse the personalBadDTD.xml.

☐ 4. Our focus will be on the `com.triveratech.training.xml.kit.saxtools.SAXTool.java` file.

☐ 5. Scroll down to the createSAXParser method and examine the code that creates and configures the parser. You will note that there is a set of statements that sets secure processing to false. This is because, with the current versions of JAXP, the secure processing defaults to true.

☐ 6. The first thing we are going to do is restore our parser to the secure processing mode. You can do this by setting the feature to true, by commenting out that entire try/catch construct, or deleting those same lines from the file. Save the file and rerun the parser in the safe more.

☐ 7. Observe how the parser behaves now. The primary difference is that the personalBadDTD.xml returns after a short period of time, objecting to more that 64,000 entity references.

☐ 8. Even 64,000 entity references still seems large. You will note that there is a statement that sets a system property limiting entity expansion to 10. Uncomment this statement, save the file, and test the new limit.

☐ 9. In those situations where you are strictly processing SOAP messages, it could be a reasonable defense to simply not allow DTDs to be parsed. There is a feature that can be set to prevent DOCTYPE statements from ever being processed. You will note the statement (a little farther down in the source code) that is currently commented out that sets the disallow DOCTYPE declaration feature true. Uncomment that line and save the file.

☐ 10. Observer how the parser behaves now. Which set of defenses would be most appropriate for a production level SOAP processor?

☐ 11. Of course, it may also be appropriate to insert some additional defenses in front of your parsing assets. Where would you place such defenses and what would they be?

Exercise 8. DYNAMIC LOADING USING XSLT

Overview	
In this exercise, you will explore invoking Java from within an XSLT stylesheet	
Objective	Understand the potential issues of unprotected stylesheets
Builds on Previous Labs	None
Time to Complete	30 minutes

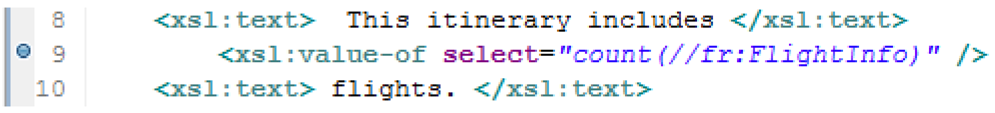
Task List

🛠 Importing and Working With XSLT

- Create a new project (General) called Stylesheet_Tutorial and import the files from **C:\StudentWork\Labs\XSL**.
- Open the **sample.xsl** file and note that XSLT editor is opened. This editor does not have a Design view but is a context sensitive editor with content assist and autocomplete.
- To execute a transform, you need to select the XML file and the stylesheet that you want run through the transformation. Holding the Control key down, right click on the schemaflightrequest.xml file and the sample.xsl stylesheet. Next, right click on one of the two selected files and select **Run As -> XSL Transformation**.
- This will invoke the XSLT Transformer and cause the output to be sent to the generated file schemaflightrequest.out.xml. Examine the output.
- Play with the stylesheet to effect the output and follow that change to the generated output.

🛠 Debugging XSLT

1. You can also debug the stylesheet, stepping through the processing, observing values and the output being incrementally built.
2. First, we need to set a breakpoint in the stylesheet so the debugging processing can be halted. Open the sample.xsl editor. On the left side of the editor there is a column between the border and the column numbers. Double-click on that column on the line of the <xsl:value-of construct. A small blue circle will appear as illustrated below:



```
    <xsl:text>  This itinerary includes </xsl:text>
        <xsl:value-of select="count(//fr:FlightInfo)" />
    <xsl:text> flights. </xsl:text>
```

3. Again, select the schemaflightrequest.xml file and the sample.xsl stylesheet. Right click on one of the two selected files and select **Debug As -> XSL Transformation**. The first thing you will encounter is a dialog box that asks if you want to debug with the Xalan 2.7.1. In order to perform debugging, you will need to use Xalan 2.5 or greater

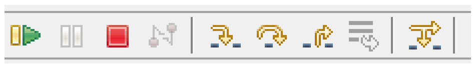
and most Java installations are not up to that level of Xalan. Your IDE comes with Xalan 2.7.1 and allows you to switch over to it to perform debugging.

Note: You can set up the debug configuration to default to run to the Xalan 2.7.1 processor, thus avoiding the dialog box during subsequent debug invocations.

4. Next, you will be asked if you want to switch to the Debug Perspective. You will want to switch to the Debug Perspective and will probably want to check to remember the decision. This will take you to the Debug Perspective and show the breakpoint in the stylesheet editor.

5. We suggest a slight reconfiguration of the perspective to take maximum advantage of what it has to offer. You will find the Result view located in the bottom window of the perspective. Drag the Result view up to the right middle tier where the Outline view resides. Now the Result view is side by side with the stylesheet editor. Note that the Result view already has some of the output generated into it.

6. At this point, the transformation is stopped at the break point you created earlier. Further steps are controlled by the icons shown below:



7. Step through the transformation watching the output being generated.

8. More complex transformations will involve stepping into deep transformations as well as the use of variables. In the debugger, the variables can be examined and even changed during the transformation.

Examine the Simple Java Call Functionality

Examine the **JavaCallSimple.xsl** file. You can run this with any XML since there are no dependencies on the input. In reviewing the stylesheet, note where the Xalan namespace is used to type the Java class being invoked. Note that we are not using a static function, but simply instantiating a Date instance and converting that to a string.

Set a breakpoint in the stylesheet and step through how the output is generated. The string that is returned is essentially what is returned by invoking the toString method on the date instance. It is possible to step into the Java from the XSLT debugging session? (No)

It is beyond the scope of this class, but you can implement and use your own Java functionality from within XSLT.

• Examine the Java Call Functionality

Examine the **JavaCall.xsl** file. You can run this with any XML since there are no dependencies on the input. In reviewing the stylesheet, note where the Xalan namespace is used to type the Java classes being invoked. Note that, since we are using two different classes, one to get the date and the second to format it, that two namespace declarations are required. This will clearly differentiate which constructs are being with Date and which are being used with the formatter.

Set a breakpoint in the stylesheet and step through how the output is generated. Note that we instantiate a date and assign the returned value to the today variable. Under the covers, the actual date object is being stored. We then instantiate an instance of the formatter, passing in a formatting pattern as an argument for the constructor. Again, the formatter instance is stored in a variable.

The last thing we want to do is call the format method on the newly created formatter, passing in the date that we previously created. The syntax for accomplishing this is definitely different than what one would expect from pure

Java. In this case, we use the formatter namespace to qualify the format method name. We pass in two arguments. This first argument is the formatter object that we want to invoke the method on. The second argument is the actual date object that is passed into the format method when the call is finally made by Xalan.

Challenge: Create a second formatter that uses the pattern: "h:mm a"
Use the both formatters with the same date object to generate the date with the first formatter and the time with the second formatter.

- **Examine the File Check Functionality**

Examine the **FileCheck.xsl** file. You can run this with any XML since there are no dependencies on the input. In reviewing the stylesheet, note where the Xalan namespace is used to type the Java class being invoked.

Set a breakpoint in the stylesheet and step through how the output is generated. Note that we instantiate a file instance and then call the static method exists, passing in the newly created file instance. As you can see, an absolute path name is provided. You will need to adjust this path name to your particular installation. Once you have set the path up correctly, trying running the transformation. It should indicate that the file was found. If not, verify the path that you entered. Now change the name of the requested file and verify that the file is not found.

What are the implications of this? What are the security implications of this? Suppose an attacker was able to modify one of your stylesheets prior to your running it in a transformation? Examine and run the **FileAttack.xsl** file to follow this line of thinking. Make sure you adjust the path so it is actually pointing at something.

One of the other implications of this is that we have a way of gracefully handling some of handling some of the file exceptions that might occur if we were using the XSLT **document()** function.

Exercise 9. UNSAFE DIRECT OBJECT REFERENCES

Overview	
Using the InsecureBank web application, you will investigate and exploit Direct Object Reference vulnerabilities. You will then identify at least two defenses that can be used to minimize or eliminate this vulnerability. Finally, you will implement an initial set of defenses to prevent these exploits.	
Objective	Gain experience recognizing and defending against unauthorized access through direct object references
Builds on Previous Labs	None
Targeted Functionality	Account operations such as balance inquiries and fund transfers
Time to Complete	60 minutes

Task List

Overview of steps:

Step 1: Setup and review InsecureBank web application
Step 2: Recognize and exploit unsafe direct object reference vulnerabilities
Step 3: Identify and implement defense against unauthorized access
Step 4: Test and extend defense

🛠 Step 1: Setup and review InsecureBank web application

☐ 1. Create a new Dynamic Web Project in your IDE targeting your runtime server. You can select a name such as **InsecureBank**

☐ 2. Import source code and the web content from **~/StudentWork/Labs/InsecureBank/Web**

☐ 3. Deploy the web application to your server and access the welcome page. You will be required to login to work with the application. Note the following as you work through this application:

- All client, authentication, and account information is mocked in the BankInfoStore class. The following datasets are established:
    - User: dan Password: idaho Accounts: 01, 02, 03
    - User: carlos Password: hiking Accounts: 04, 05, 06
    - User: jane Password: pain Accounts: 07, 08, 09
    - All accounts are typed (checking, savings, or DID savings)
    - All accounts start with a balance of $1000
- Since data is mocked, no database is required and data changes will not persist across application restarts.
- All functionality requires authentication (or at least that is the intent).

- There are several vulnerabilities and weaknesses to the authentication portion of this application. However, those are not the focus of this lab. If you have time and are so inclined, feel free to identify those vulnerabilities. Your instructor can review them after the lab.
- There are basically three operations that an authenticated user can perform:
  - Check account balances
  - Transfer funds between accounts (constrained by available funds)
  - Transfer funds from an account to a credit card (used to either pay off the balance of the card or to add more cash to the account for future use).
- Normally, authentication challenges and subsequent operations would be performed within an SSL connection. We are not going to set those up since all vulnerabilities we are looking at can occur within an existing SSL session.

🛠 Step 2: Recognize and exploit unsafe direct object reference vulnerabilities

☐ 1. Work through each of the account operations. Feel free to use Fiddler to capture all of the requests and responses. Since all operations use Get requests, examining the URLs can be nearly as useful. If operations were using Post requests, a tool such a Fiddler would be required to examine traffic.

☐ 2. Identify Direct Object Reference issues.

☐ 3. Verify that significant exploits can occur.

🛠 Step 3: Identify and implement defense against unauthorized access

☐ 1. Based on the presentation materials and associated discussions identify at least two defenses that could be used to minimize or eliminate direct object references and the associated unauthorized access of accounts.

☐ 2. Design and implement one of the defenses.

🛠 Step 4: Test and extend defense

☐ 1. Test your defense and verify that it precludes unauthorized access of resources

☐ 2. If you have time, design, implement, and test the second defense that you identified in the previous step.

☐ 3. Questions to consider:

a. Which defense is easier to implement?

b. Which defense is more effective?

c. Ultimately, is either defense adequate by itself?

  i. Why not?

Exercise 10. Defending Against XSS

Overview	
During this exercise, you will explore XSS vulnerabilities. You will try various attacks against the application. You will then implement defenses against these attacks.	
Objective	Gain experience recognizing and defending against XSS.
Builds on Previous Labs	`InsecureWeb Project`
Targeted Files	EmailUpdateController.java web.xml
Time to Complete	60 minutes

Task List

Overview of steps:

Step 1: Explore the XSS attacks
Step 2: Design and implement an effective XSS defense
Step 3: Test your defense

🛠 Step 1: Explore XSS attacks

☐ 1. For this lab, we will be focusing on the email update functionality in our case study. Examine the functionality associated with the **Update Email** functionality. As you follow the thread of execution, you will see that the updated email is persisted to the HSQL database. Test the functionality, using the **Query for Email** link to verify that any email changes are persisted.

☐ 2. Try various strings in the email address field and see how they are processed in the response to the query. You are inserting the updated email strings into a database. At a later time, using the query link, you are extracting those same string and placing them into an executable context.

☐ 3. The following values (inserted as email addresses) illustrate examples of fairly basic XSS attacks (these can be found in electronic form in the **XSSExamples.txt** file in the **StudentWork/Labs/XSS** folder.):

```
<b>XSS Attack!</b>
<a href=http://www.cnn.com>View Here!</a>
<script>alert("XSS")</script>
<script>document.write("<b>Inject code here!</b>")</script>
<script>document.write(document.cookie)</script>
<img src="http://none" onerror="alert(document.cookie);">
<img src=http://none onerror=parent.location="http://localhost:8080";>
```

☐ 4. Try entering each of these in conjunction with a username and then querying them to extract the inserted values and place them into an HTML page. Note that same examples may not completely work if you do not have Internet access. In that case, you can enter a value of http://localhost:8080 to see how your browser is taken to another website.

🛠 Step 2: Design and implement an effective XSS defense

☐ 1. Develop rigorous, positive specifications for the trust boundaries associated with the email update functionality.

☐ 2. Implement defenses that are strong enough to effectively defend against active code and malicious scripts. (Do not implement any defenses for the email query functionality…we will be exploring that in the next lab.)

🛠 Step 3: Test your defenses

☐ 1. Test your defenses for the email update functionality.

☐ 2. It is a common assumption that because a database resides behind firewalls and we have physical control of that database, that its content can be trusted. Spend a few minutes analyzing whether you would trust this database's content. Should you establish a trust boundary with your database? If so, how would you defend what would now be considered unvalidated input.

☐ 3. Determine if your previous defenses that you have built are vulnerable to XSS attacks. If you are so inclined, deactivate some of your defenses and attack these other trust boundaries. Then reactivate your defenses and verify that they are working.

Exercise 11. SPOTLIGHT: EQUIFAX

Overview	
During this exercise, you will investigate reports concerning the Equifax exploit.	
Objective	Learn that any incident is the result of multiple failures.
Builds on Previous Labs	None
Targeted Files	None
Time to Complete	45 minutes

Task List

Overview of Steps:

Step 1: Investigate the Equifax exploit
Step 2: Discuss your conclusions with classmate/co-worker

🛠 Step 1: Investigate the Equifax exploit

☐ 2. Your task will be to review various sources discussing the recent Equifax exploit.

☐ 3. If you have internet access, use the search engines to dig up some information.

a. We have also provided a couple of sources in the lab folder for this lab.

☐ 4. Determine

a. What is the impact of the exploit?

b. What were the mechanisms used to manifest the exploit?

c. Try and identify the stages of the exploit in terms of infiltration, propagation, aggration, and exfiltration.

d. How the exploit detected?

e. What were the defensive failures that contributed to the exploit?

🛠 Step 2: Discuss your conclusions with classmate/co-worker

☐ 6. Discuss the results of your investigation with a classmate or a co-worker.

☐ 7. Examine

a. The similarities and differences in your conclusions as well as the conclusions of the sources that you analyzed.

b. How the information about the exploit relates to what has been discussed in this class.

c. How developers could have applied defensive measures to help prevent or mitigate the impact of this exploit.

d. How AppSec teams could have helped to defend against this exploit.

☐ 8. Prepare to discuss your conclusions with the rest of the class.

Exercise 12. ERROR HANDLING

Overview	
Working in a team, you will analyze the error handling in the case study.	
Objective	Experience critical thinking about error handling.
Builds on Previous Labs	`InsecureWeb`
Approximate time	40 minutes

Task List

Overview:

Step 1: Regroup and analyze error handling
Step 2: Review error handling analysis as class

🛠 Step 1: Regroup and analyze error handling

☐ 1. Students should return to the groups formed earlier in the class.

☐ 2. From an overall project perspective, we would like to perform a comprehensive error handling review. However, we probably do not have the time for that. Select one of the functional areas of the InsecureWeb application (as delineated by the home page for the web application).

☐ 3. Identify the following, based on the information in the previous unit:

a. What are error states have been identified, and how are those errors handled?
b. What error states remain to be handled?
c. Is an appropriate level of information being provided in error pages or messages?
d. Are there any unhandled exceptions?

☐ 4. Spend about 15-20 minutes on this activity. If possible, review a more than one portion of the application.

🛠 Step 2: Review error handling analysis as class

☐ 1. Once the class regroups, the instructor will lead a discussion of each group's results.

Exercise 13. CROSS-SITE REQUEST FORGERIES

Overview	
During this exercise, you will examine the credit card transfer functionality. This functionality is used in the context of an authenticated session. You will attack web applications with and without Direct Object References defenses in place. Despite these defenses, they are vulnerable to CSRF attacks. You will prove that such exploits can occur and see the impact of such an attack. You will then identify at least two defenses that have been used in the past and proven not to be effective. Finally, you will implement an initial set of defenses to prevent these exploits.	
Objective	Gain experience recognizing and defending against unauthorized access through CSRF attacks
Builds on Previous Labs	`InsecureBank Project`
Targeted Functionality	Transfer to Credit Card
Time to Complete	60 minutes

Task List

Overview of Steps:

Step 1: Recognize and exploit unsafe CSRF vulnerabilities
Step 2: Analyze various potential CSRF defenses
Step 3: Identify and implement defense against CSRF
Step 4: Test defense

🛠 Step 1: Recognize and exploit unsafe CSRF vulnerabilities

Note: You will need to use the same browser for all of the operations in this lab. Make sure you understand why that is. The reason is central to the CSRF exploit.

☐ 5. Create a new Dynamic Web Project in your IDE targeting your runtime server. You can select a name such as **TestBank**

☐ 6. Import source code and the web content from **~/StudentWork/Labs/InsecureBankCSRF/Web**

☐ 7. Deploy the web application to your server and access the welcome page. You will be required to login to work with the application.

☐ 8. Examine the Credit Card Transfer functionality. Note that the functionality requires that a valid authentication session be in place.

☐ 9. Feel free to use Fiddler to capture all of the requests and responses. Since all operations use Get requests, examining the URLs can be nearly as useful. If operations were using Post requests, a tool such a Fiddler would be required to examine traffic.

☐ 10. Capture an example of a URL that would transfer funding from an account to a credit card.

☐ 11. How could you get a user to submit a similar URL (that has a temporary credit card under your control) during an active authentication session?

☐ 12. Open and examine the test.html file that you will find in the **~/StudentWork/Labs/InsecureBankCSRF** folder. Note the URL and modify it to match the URL you captured from your application. Save the file.

☐ 13. Make sure that you have an open, authenticated session in your browser. Set your browser up to have the account operations page open.

☐ 14. In another tab, open the test.html page.

☐ 15. Note the lack of activity in either tab.

☐ 16. Check the balance of the account that you referenced in the test.html file.

☐ 17. Refresh the test.html page several times and check the balance again. You should see a steadily diminishing balance in the account (and, presumably, a steadily increasing balance on the credit card).

☐ 18. Adjust the URL in the test.html to point to your **InsecureBank** web application. Login into the **InsecureBank** and repeat the CSRF attack against this web application. You should see the exploits happening even with your Direct Object Access defenses in place.

🛠 Step 2: Analyse various potential CSRF defenses

☐ 9. Now that you understand and see the mechanism by which a CSRF attack can occur, consider the various ineffective defenses that were referenced in the presentation materials.

☐ 10. Make sure that you understand why those defenses are ineffective. Implement some of them to prove to yourself that is the case

🛠 Step 3: Identify and implement defense against CSRF

☐ 1. Based on the presentation materials and associated discussions identify a defense that could be used to minimize or eliminate CSRF and the associated unauthorized access of accounts.

☐ 2. Design and implement the defense.

🛠 Step 4: Test the defense

☐ 1. Test your defense and verify that it precludes unauthorized access of resources.

☐ 2. Make sure that you understand the mechanics of the defense.

Exercise 14. REVIEW OF RECENT INCIDENTS

Overview	
This exercise is simply a reading and analysis assignment. As you examine these incident reports consider their implications relative to the software and applications that you work on.	
Objective	Gain an appreciation for the relevance of what this course is about.
Builds on Previous Labs	None
Time to Complete	20 minutes

Task List

Overview of Steps:
Step 1: Review incident reports

🛠 Step 1: Review incident reports

1. **<u>Facebook/Adobe/Password:</u>** Facebook is using a list of hacked Adobe accounts posted by the miscreants themselves to warn its own customers about password reuse.

   The social network mined data leaked as the result of the recent breach at Adobe in an effort to provide timely warnings and prompt its users to secure their accounts. Facebook users who used the same email and password combinations on Adobe are required to both change their password and answer additional security questions, investigative reporter Brian Krebs reports.
   ...
   Adobe original said hackers had stolen nearly three million customer credit card records, as well as undetermined volume of user accounts login credentials. The software firm later admitted that the encrypted account data of 38 million users had leaked.

   But when a dump of the offending customer database appeared online it contained not online just 38 million, but 150 million credentials. Leaked information includes internal ID, user name, email, encrypted password and password hints.

   That alone would be bad enough but Adobe compounded the problem by failing to follow industry best practices about only stored passwords credentials as properly salted hashes.

   In particular, Adobe erred in using a single encryption key to encrypt user credentials, as explained in some depth by security veteran Paul Ducklin in a post on Sophos's Naked Security blog. Security researchers have figured out a substantial proportion of the leaked user passwords using a variety of inferences, such as leaked data from other large password breaches.

   For example, security researcher Jeremi Gosney of the Stricture Group came across the purloined passwords on one of several online dumps before analysing them to see which passwords are most-used by Adobe customers.

   The resulting list of the top 100 most commonly used passwords in the Adobe dump is full of FAIL. "123456" and (of course) "password" are in the top three of the rest are hardly any better.

Gosney worked out that a whopping 1.9 million of Adobe's customers use the string “123456” as their password. We can only hope that the majority of such users didn't reuse these passwords elsewhere on more sensitive sites, such as e-banking, social networking and webmail. It could be that some people who didn't really care about their Adobe account were more careful elsewhere.
http://www.theregister.co.uk/2013/11/14/facebook_adobe_password_leak_warning/

2. **ColdFusion/Hacktivists:** Hacktivists allegedly affiliated with Anonymous have been covertly breaking into US government systems and pilfering sensitive information for nearly a year, the FBI warned last week.

   The attacks (which began last December and are thought to be ongoing) exploit flaws in Adobe's ColdFusion web app development software to plant backdoors on compromised systems, according to an FBI memo seen by Reuters. The memo said the US army, Department of Energy, Department of Health and Human Services, and others had all been targeted.

   Officials told the news agency that the warning was linked to attacks allegedly carried out by Lauri Love, 28, of Stradishall, England and others. Love alone was indicted in New Jersey last month over a string of attacks that matches that latest warnings.

   A DoJ statement on the indictment lists 10 attacks against US government systems, eight of which are blamed on ColdFusion exploits1. The remaining two attacks were blamed on SQL injection-style assaults.
   http://www.theregister.co.uk/2013/11/18/anon_us_gov_hack_warning/

3. **Ransomware:** Massachusetts cops have admitted paying a ransom to get their data back on an official police computer infected with the devilish Cryptolocker ransomware.

   Cryptolocker is a rather unpleasant strain of malware, first spotted in August, that encrypts documents on the infiltrated Windows PC and will throw away the decryption key unless a ransom is paid before a time limit. The sophisticated software, which uses virtually unbreakable 256-bit AES and 2048-bit RSA encryption, even offers a payment plan for victims who have trouble forking out the two Bitcoins (right now $1,200) required to recover the obfuscated data.

   On November 6, a police computer in the town of Swansea, Massachusetts, was infected by the malware, and the cops called in the FBI to investigate. However, in order to get access to the system the baffled coppers decided that it would be easier to pay the ransom of 2 BTC, then worth around $750, and received the private key to unlock the computer's data on November 10.

   "It was an education for [those who] had to deal with it," Swansea police lieutenant Gregory Ryan told the Herald News. "The virus is so complicated and successful that you have to buy these Bitcoins, which we had never heard of."

   Ryan said that essential police systems weren't affected by the infection, and federal agents are still investigating the infection, hopefully to find clues that'll lead the Feds to the malware's writer. The software nasty is thought to have been the work of Eastern European criminal gangs, but no one knows for sure.

   "The virus is not here anymore," Ryan said. "We've upgraded our antivirus software. We're going to try to tighten the belt, and have experts come in, but as all computer experts say, there is no foolproof way to lock your system down."

   Apart from not being a fool that is. Cryptolocker primarily spreads via email attachments, typically a PDF that claims to be from a government department or delivery service. As ever, experts advise not to open attachments unless you are sure of its contents and the source
   http://www.theregister.co.uk/2013/11/21/police_pay_cryptolocker_crooks_to_get_their_computers_back/

4. **Linux:** Announced February 21, 2016. Title: Linux Mint Website Compromised, Found to Redirect Users to ISOs Containing Malware

Description: The Linux Mint Project has disclosed that over the weekend its website was compromised and redirected users who were looking to download a Linux Mint 17.3 Cinnamon ISO to one containing a backdoor.

The Linux Mint Project has also disclosed that forums were also compromised and attackers potentially stole usernames, email addresses, encrypted copies of passwords, and other personal information that a user might have put into their forum profile. Note that users who download Linux Mint 17.3 Cinnamon via a direct HTTP link or via torrents are unaffected. Researchers have indicated that the compromised ISO contains malware identified as Tsunami. The Linux Mint website has been fixed and no longer point users to the malicious ISO.

5. **Java 6 and Xerces:** Apache Xerces2 Java, as used in Sun Java Runtime Environment (JRE) in JDK and JRE 6 before Update 15 and JDK and JRE 5.0 before Update 20, and in other products, allows remote attackers to cause a denial of service (infinite loop and application hang) via malformed XML input, as demonstrated by the Codenomicon XML fuzzing framework. Original release data of 08/06/2009. CVE-2009-2625

6. **Multiple Programming Language Implementations Vulnerable to Hash Table Collision Attacks**
(December 28, 2011 at 01:04 pm) US-CERT is aware of reports stating that multiple programming language implementations, including web platforms, are vulnerable to hash table collision attacks. This vulnerability could be used by an attacker to launch a denial-of-service attack against websites using affected products.

   The Ruby Security Team has updated Ruby 1.8.7. The Ruby 1.9 series is not affected by this attack. Additional information can be found in the ruby 1.8.7 patchlevel 357 release notes.

   Microsoft has released an update for the .NET Framework to address this vulnerability and three others. Additional information can be found in Microsoft Security Bulletin MS11-100 and Microsoft Security Advisory 2659883.
   Some programming language implementations do not sufficiently randomize their hash functions or provide means to limit key collision attacks, which can be leveraged by an unauthenticated attacker to cause a denial-of-service (DoS) condition.
   I. Description
   Many applications, including common web framework implementations, use hash tables to map key values to associated entries. If the hash table contains entries for different keys that map to the same hash value, a hash collision occurs and additional processing is required to determine which entry is appropriate for the key. If an attacker can generate many requests containing colliding key values, an application performing the hash table lookup may enter a denial of service condition.
   Hash collision denial-of-service attacks were first detailed in 2003, but recent research details how these attacks apply to modern language hash table implementations.
   II. Impact
   An application can be forced into a denial-of-service condition. In the case of some web application servers, specially-crafted POST form data may result in a denial-of-service.
   III. Solution
   Apply an update
   Please review the Vendor Information section of this document for vendor-specific patch and workaround details.

   Limit CPU time
   Limiting the processing time for a single request can help minimize the impact of malicious requests.

   Limit maximum POST size
   Limiting the maximum POST request size can reduce the number of possible predictable collisions, thus reducing the impact of an attack.

   Limit maximum request parameters
   Some servers offer the option to limit the number of parameters per request, which can also minimize impact.

7. **JBoss:** JBoss sysadmins need to get busy hardening their systems, with a rising number of attacks against the system, according to Imperva.

   The attacks are based on an exploit that was published back in October by Andrea Micalizzi. The exploit code gave remote attackers arbitrary code execution access to HP's PCM Plus and Application Lifecycle Management systems without authentication.

   The attack also works against McAfee, Symantec and IBM products using JBoss 4.x and 5.x.

   Imperva's advisory states that the compay is now seeing an increasing amount of attack traffic using the exploit.

   What's surprising, Imperva says, is that while the Micalizzi exploit code only hit the waiting world this year, the vulnerability has been known since 2011. The attack works by exploiting the HTTP invoker service in JBoss, used to provide access to Enterprise Java Beans.

   Imperva says the Micalizzi exploit "abuses invoker/EJBInvokerServlet to deploy a web shell code that enables the hacker to execute arbitrary Operating System commands on the victim sever's system." In the HP environment, this would provide access to the PCM Plus and ALM management consoles.

   There are currently 23,000 servers exposing their JBoss management interfaces to the Internet, up from 7,000 in 2011, Imperva says, with infections spotted in the wild.
   http://www.theregister.co.uk/2013/11/19/old_jboss_vuln_in_the_wild_needs_patching/

8. **SAP BusinessObjects Axis2 Default Admin Password, 2011** The Axis2 component of SAP BusinessObjects contains a default administrator account and password.

   I. Description
   The SAP BusinessObjects product contains a module (dswsbobje.war) which deploys Axis2 with an administrator account which is configured with a static password. As a result, anyone with access to the Axis2 port can gain full access to the machine via arbitrary remote code execution. This requires the attacker to upload a malicious web service and to restart the instance of your server. This issue may apply to other products and vendors that embed the Axis2 component. The username is "admin" and the password is "axis2", this is also the default for standalone Axis2 installations. Additional details may be found in the Rapid7 R7-0037 Advisory.
   II. Impact
   An attacker can execute arbitrary code by creating a malicious web service (jar). The attacker can log in to the Axis2 component with the default admin account, upload the malicious web service, and upon restart the malicious code will be executed.
   III. Solution
   The vendor has addressed this vulnerability in SAP Security Note 1432881.
   Users should change the admin default password. This can be done by modifying the password value within axis2.xml

Exercise 15. SPOTLIGHT: CAPITAL ONE

Overview	
During this exercise, you will investigate reports concerning the recent Capital One exploit.	
Objective	Learn that any incident is the result of multiple failures.
Builds on Previous Labs	None
Targeted Files	None
Time to Complete	30 minutes

Task List

Overview of Steps:

Step 1: Investigate the Capital One exploit
Step 2: Discuss your conclusions with classmate/co-worker

🛠 Step 1: Investigate the Capital One exploit

☐ 19. Your task will be to review various sources discussing the recent Capital One exploit.

☐ 20. If you have internet access, use the search engines to dig up some information.

a. We have also provided a couple of sources in the lab folder for this lab.

☐ 21. Determine

a. What is the impact of the exploit?

b. What were the mechanisms used to manifest the exploit?

c. Try and identify the stages of the exploit in terms of infiltration, propagation, aggration, and exfiltration.

d. How the exploit detected?

e. What were the defensive failures that contributed to the exploit?

🛠 Step 2: Discuss your conclusions with classmate/co-worker

☐ 11. Discuss the results of your investigation with a classmate or a co-worker.

☐ 12. Examine

a. The similarities and differences in your conclusions as well as the conclusions of the sources that you analyzed.

b. How the information about the exploit relates to what has been discussed in this class.

c. How developers could have applied defensive measures to help prevent or mitigate the impact of this exploit.

d. How AppSec teams could have helped to defend against this exploit.

☐ 13. Prepare to discuss your conclusions with the rest of the class.

Exercise 16. WEB SERVICE ATTACKS

Overview	
In this exercise, you will deploy a web service that performs email management services such as email queries and email updates using SOAP-based web service interactions over HTTP. You will then test the web service and then attack it using many of the attacks that we have covered in the class.	
Objective	Examine how attacks can be mounted against web services.
Files	
Classes	
Approximate time	60 minutes

Tasks

🛠 Step 1: Review the background materials

Thus far, we have interacted with the Case Study using web applications. Now we are going to shift our focus to interacting with the application using a web service. In this case, we are focusing on querying and updating email addresses based on usernames.

The web service and a rudimentary client have already been implemented for you. As you import and deploy the service, take the time to examine its functionality and test the deployed service using the Web Service Explorer.

Once you have worked with the web service to your satisfaction, you will import a simple Java application that can be used to open a socket to the web service and submit SOAP requests. We have provided various SOAP requests that you can use to query and update the email as well as mount SQL Injection, XSS, and denial of service attacks.

🛠 Step 2: Create a Web project and setup the Email Management Service

Create a Dynamic Web Project called **WSAttacks**

Import the resources for the **WSAttacks** project from the **~/StudentWork/Labs/WebServiceAttacks/Web** folder. Import all of the code from the **src** folder into the **WSAttacks/src** folder, and all the code from **WebContent** into the **WSAttacks/WebContent** folder.

🛠 Step 3: Familiarize yourself with the files

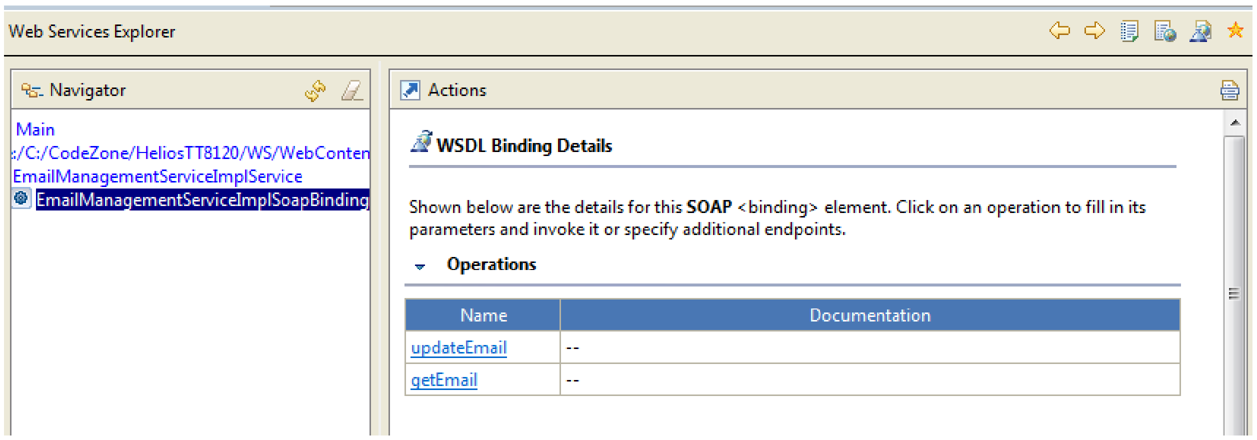
The web service is implemented in the **EmailManagementServiceImpl** class. If you examine this, you will find that there are two methods supported – **getEmail** and **updateEmail**.

The web service stack used for this service is Apache Axis using JAX-RPC. You will find the required libraries for this in **WEB-INF/lib** folder. The WSDL for the service can be found in **WebContent/wsdl.**

🛠 Step 4: Deploy and Test the service

Deploy the **WSAttacks** project to the server, restarting the server if required. You will also need to make sure that the database is running. We are also assuming that you have deployed the **InsecureWeb** application and it is running.

Once the **WSAttacks** project is properly deployed, right click on the web service's WSDL and select **Web Services -> Test with Web Service Explorer**. This will start the Web Services Explorer and open a basic testing interface that you can use to test the newly deployed service.


Web Services Explorer
Navigator
Main
:/C:/CodeZone/HeliosTT8120/WS/WebConten
EmailManagementServiceImplService
EmailManagementServiceImplSoapBinding
Actions
WSDL Binding Details

Shown below are the details for this SOAP <binding> element. Click on an operation to fill in its parameters and invoke it or specify additional endpoints.

Operations

Name	Documentation
updateEmail	--
getEmail	--

Double-click on the **getEmail** operation, enter a valid username and ensure that you get an expected response. Use the **updateEmail** operation and verify that it is also working correctly.

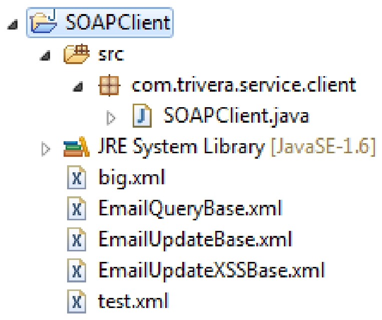
At this point, you have successfully deployed a web service and tested its two operations. Leave the service deployed and the server running.

🛠 Step 5: Setup a service client

The client that we will use is a Java application that runs from the command line. This will require creating a Java project, importing code and resources, and launching the application using a run configuration.

First, switch to the **Java** perspective and create a new Java project called **SOAPClient**.

Now import the Java code for the project from the Java location at **~/StudentWork/Labs/WebServiceAttacks/Java.** Import all of the code from the **src** folder into the **SOAPClient/src** folder, and all the resources from the **resources** folder into the **SOAPClient** base folder. The project should look something like this:



SOAPClient is a general-purpose SOAP message sender class has been provided for you. The **SOAPClient** application takes two command line arguments. The first argument is the location (endpoint) of the web service. The second argument is the XML file that is the

SOAP message to be submitted to the web service. Take a minute and examine the source code for **SOAPClient**.

Note the **test.xml** file. This is an example of a file that might contain a SOAP request to be sent using this Java application.

🛠 Step 6: Programmatically Use the Web Service

In order to run the **SOAPClient** with command line arguments, we have to setup a run configuration that we can use to enter the arguments. Right-click on the **SOAPClient** file and select **Run As -> Run Configuration**. This will bring up a dialog box with a variety of potential run configurations. Select **Java Application** and then create a new one that will run **SOAPClient** as the main class. Now select the **Arguments** tab and enter the required arguments into the **Program arguments** window. For example:
http://localhost:8080/WSAttacks/services/EmailManagementServiceImpl test.xml

Note that the URL is the endpoint that is in the WSDL for the project. Select **Apply** and then **Run**.

There will be two consoles in the IDE, one for the server and one for the Java application that you just launched and ran. Select the terminated Java console and you should see an expected SOAP response containing the correct value as the payload.

Once you have successfully queried the service, use the update operation to change an email address and verify a successful update.

🛠 Step 7: Perform XSS attacks via the web service

You will need to use the update operation to insert your XSS payload into the database. Since you are working in XML, the XSS payload will need to be carried in a CDATA section. An example is provided for you in the **EmailUpdateXSSBase.xml** file. Feel free to try all of the XSS attacks that we explored in our previous XSS lab. Of course, the best way to test these attacks is to use the query capability in a browser using the **CaseStudy** web application.

Once you have completed your attacks, you can reset your database if you feel so inclined.

🛠 Step 8: Perform SQL Injection attacks via the web service

In this case, you can use the **getEmail** operation to mount your SQL Injection attacks. SQL injection attacks usually have less sensitivity relative to XML so you should not have to wrap the SQL Injection payload in a CDATA section. Feel free to try all of the attacks that we explored in our previous SQL Injection lab.

🛠 Step 9: Perform DoS attack via the web service

In this case, you can use the **big.xml** file to mount your DoS attack. Note that this XML is a completely wrong set of tags and namespace for the web service. It is also extremely large and should cause, at a minimum, a significant pause…perhaps even an out of memory exception.