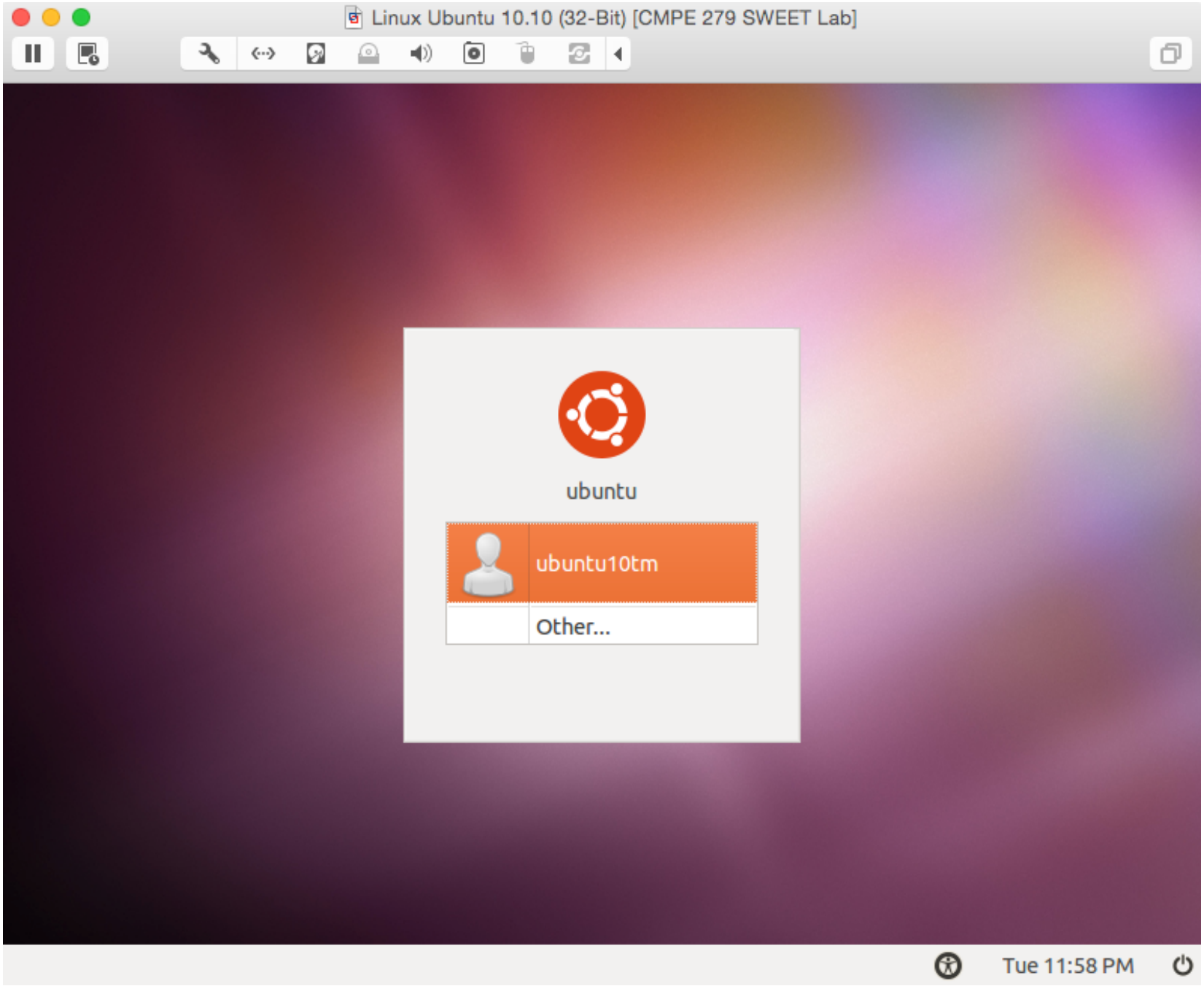
Software Security

Module 8 - JSP, MySQL & Java Security

CMPE279
Software Security Technologies
San Jose State University

GIT on Lab VM


Linux Ubuntu 10.10 (32-Bit) [CMPE 279 SWEET Lab]
ubuntu
ubuntu10tm
Other...
Tue 11:58 PM

Ubuntu Sweet VM

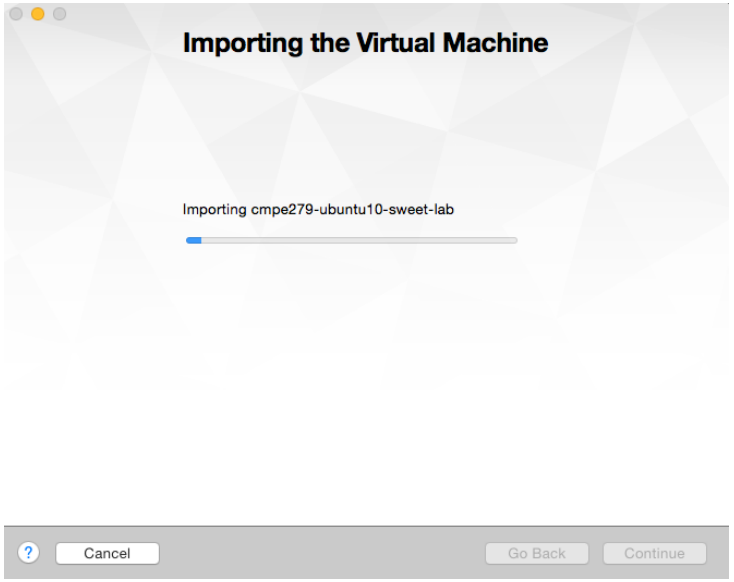
user login: user (ubuntu10tm)
user pass: 123456

root login: root
root pass: 123456

~~**GitHub/SVN Checkout:**~~

~~cd ~/CMPE279~~
~~svn co https://github.com/paulnguyen/cmpe279/trunk .~~

~~(to update, type "***svn update***" in CMPE279)~~



Tomcat 7

admin user: tomcat
password: tomcat

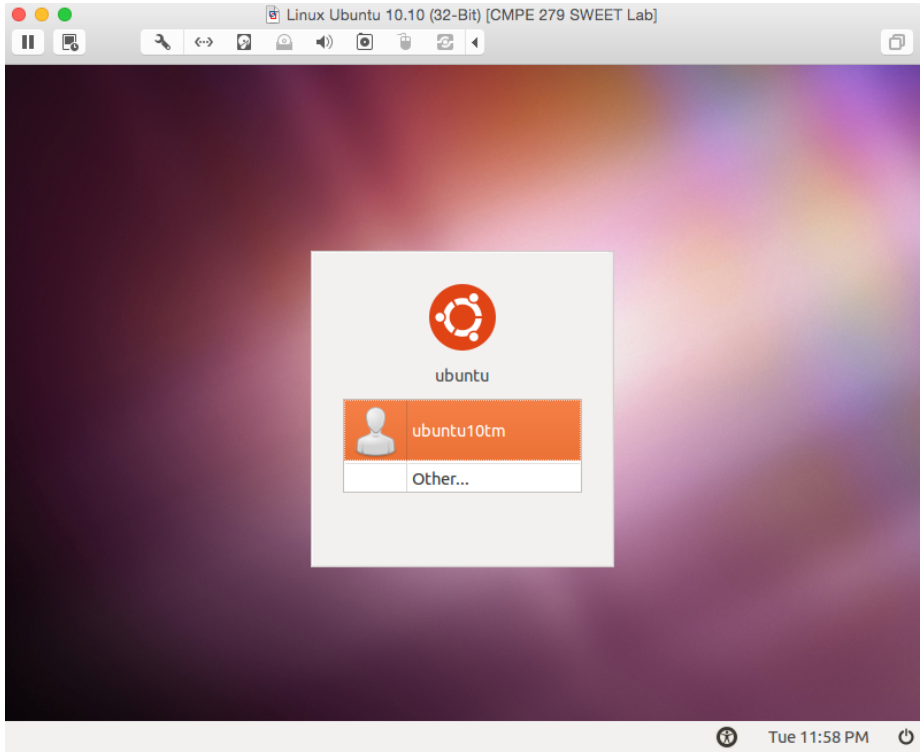
Tomcat 6 (WebGoat 5.3)

admin user: tomcat / tomcat
webgoat admin: webgoat / webgoat
webgoat basic: basic / basic
web goat guest: guest / guest

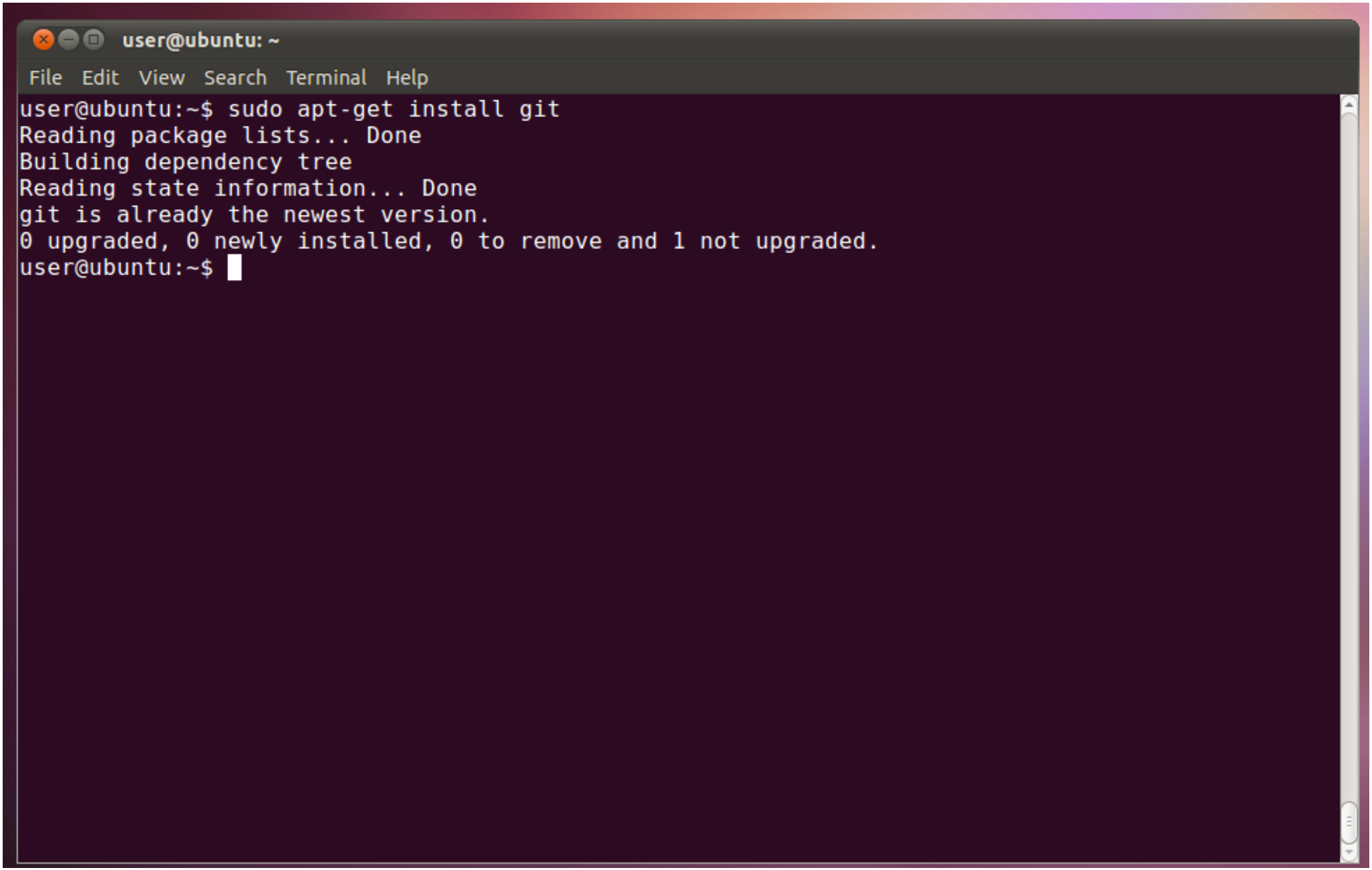
MySQL 5.1.49

root user: root / 123456

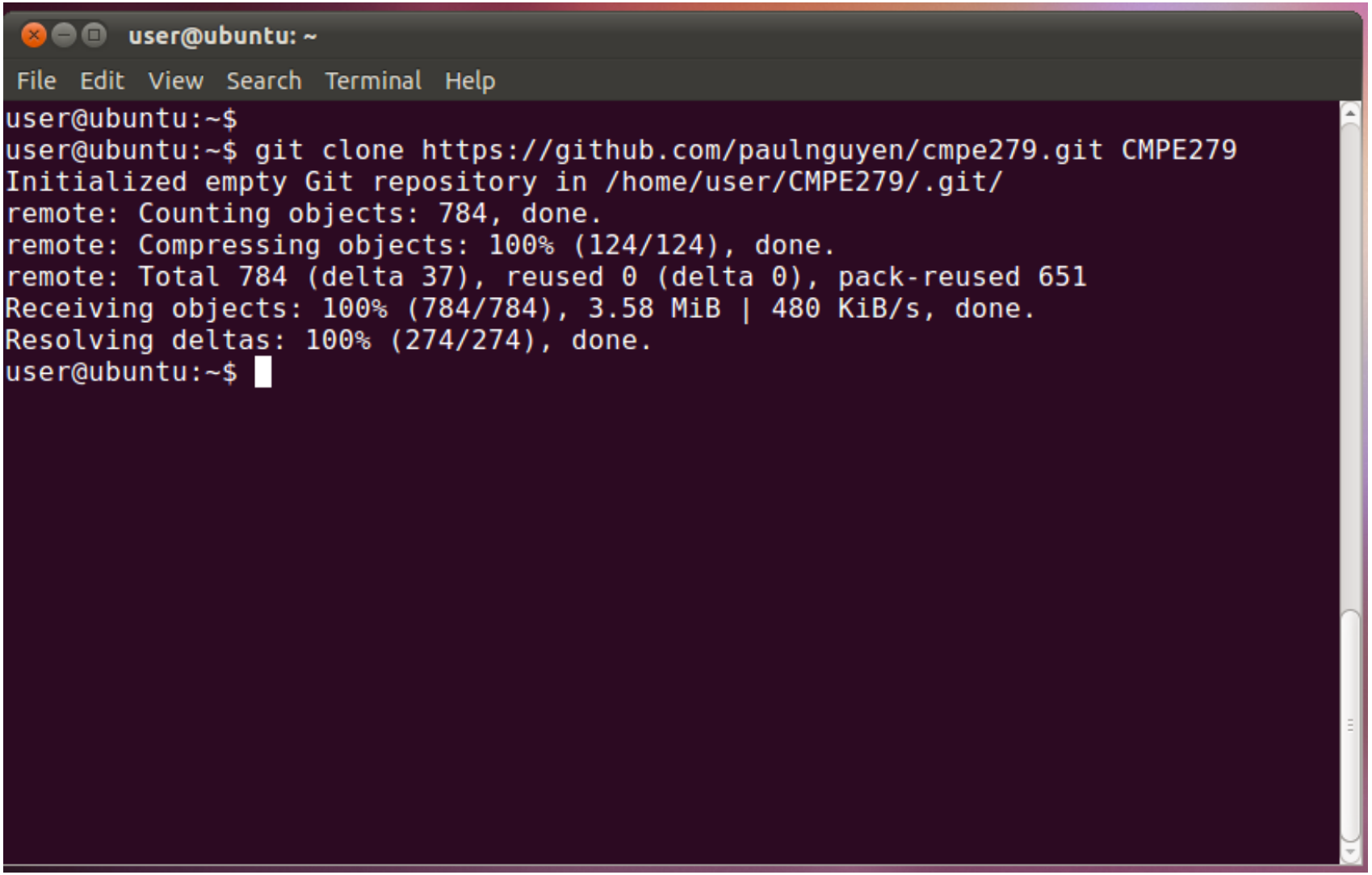
databases: badstoredb, mysql, test



Install GIT Command Line (if missing)



In home dir, clone github project



GIT COMMANDS

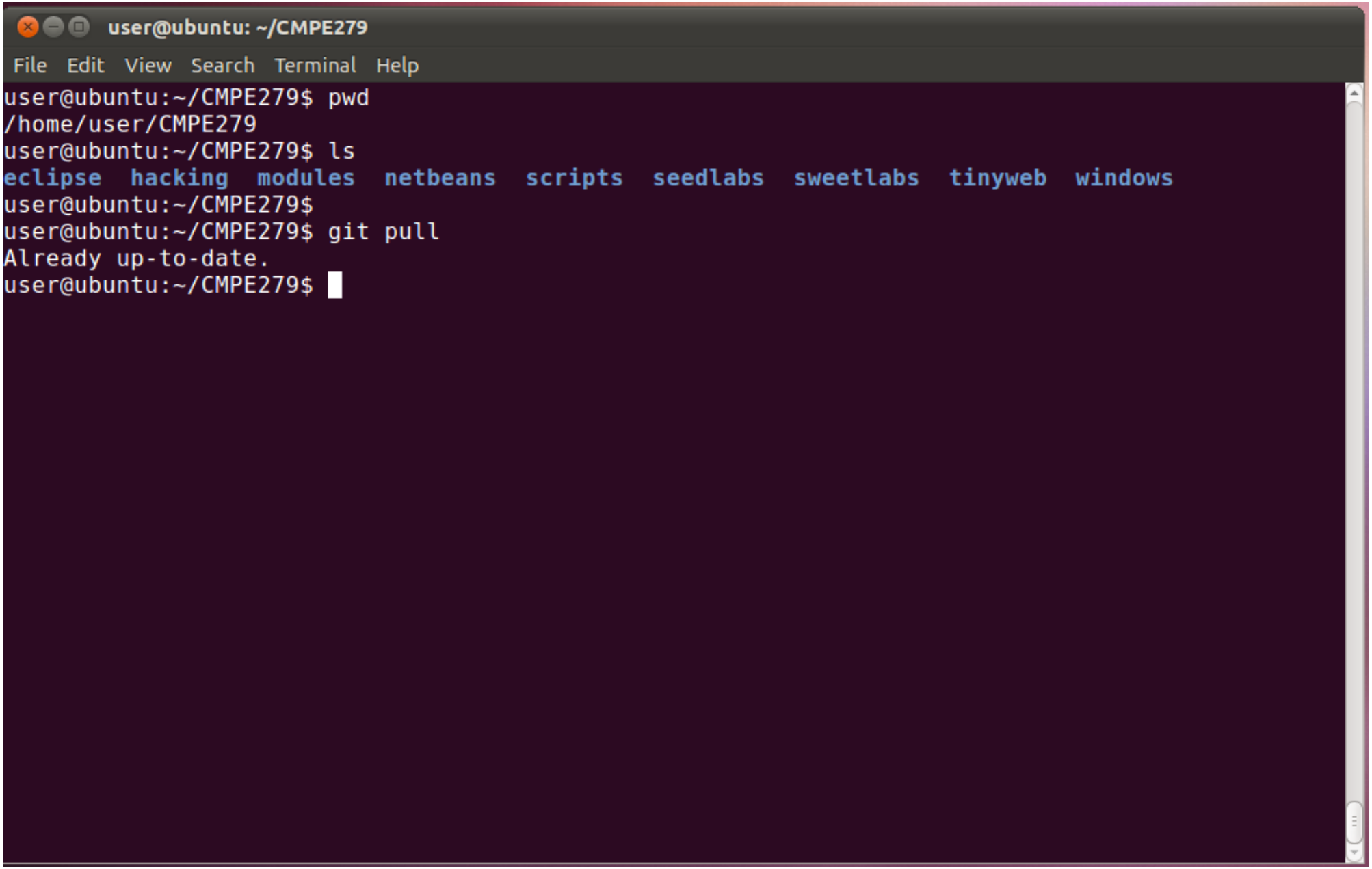
Initial or Clean Checkout:

cd ~
git clone https://github.com/paulnguyen/cmpe279.git CMPE279

Pulling Updates:

cd ~/CMPE279
git pull

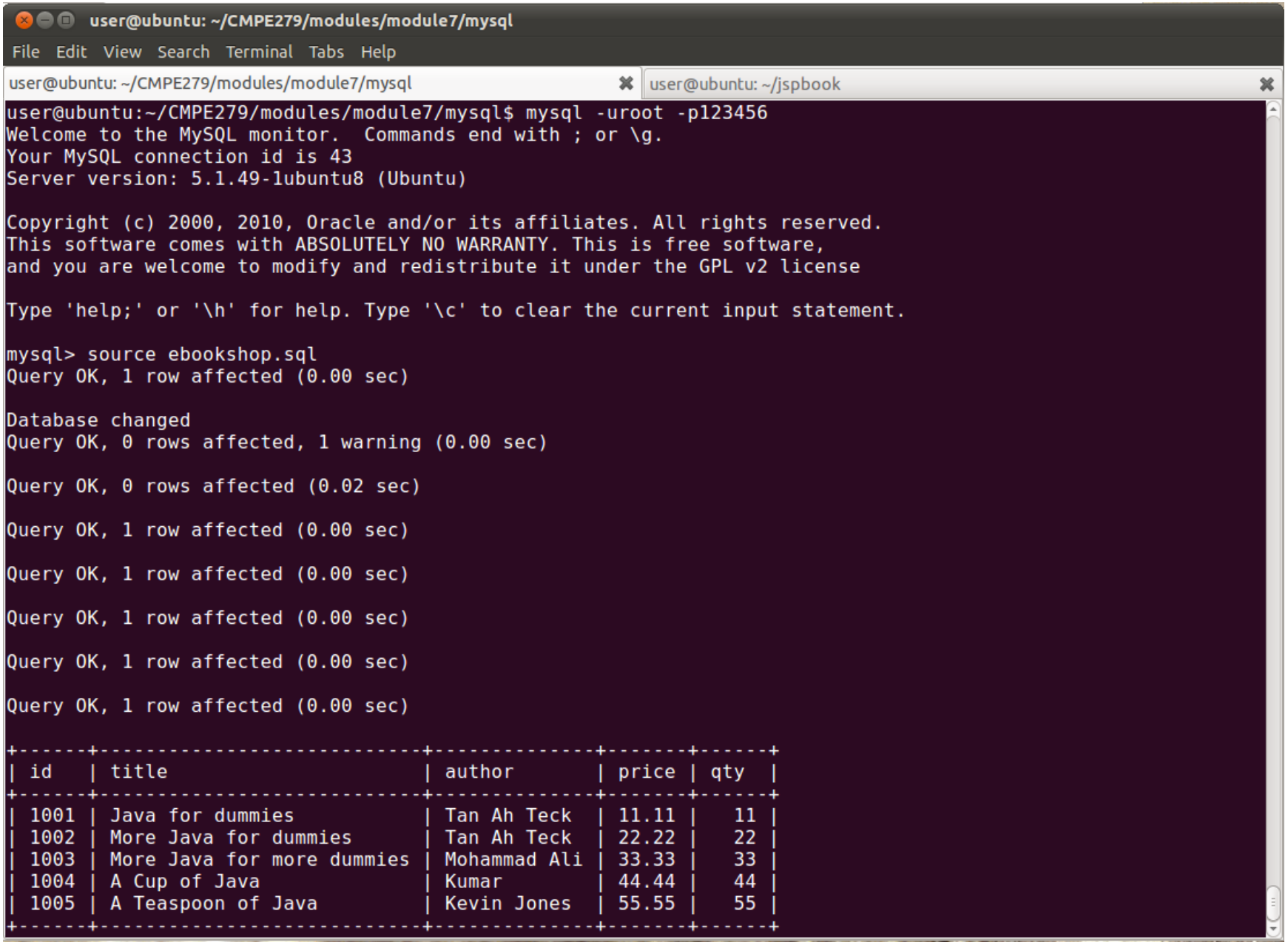
Pulling updates from github



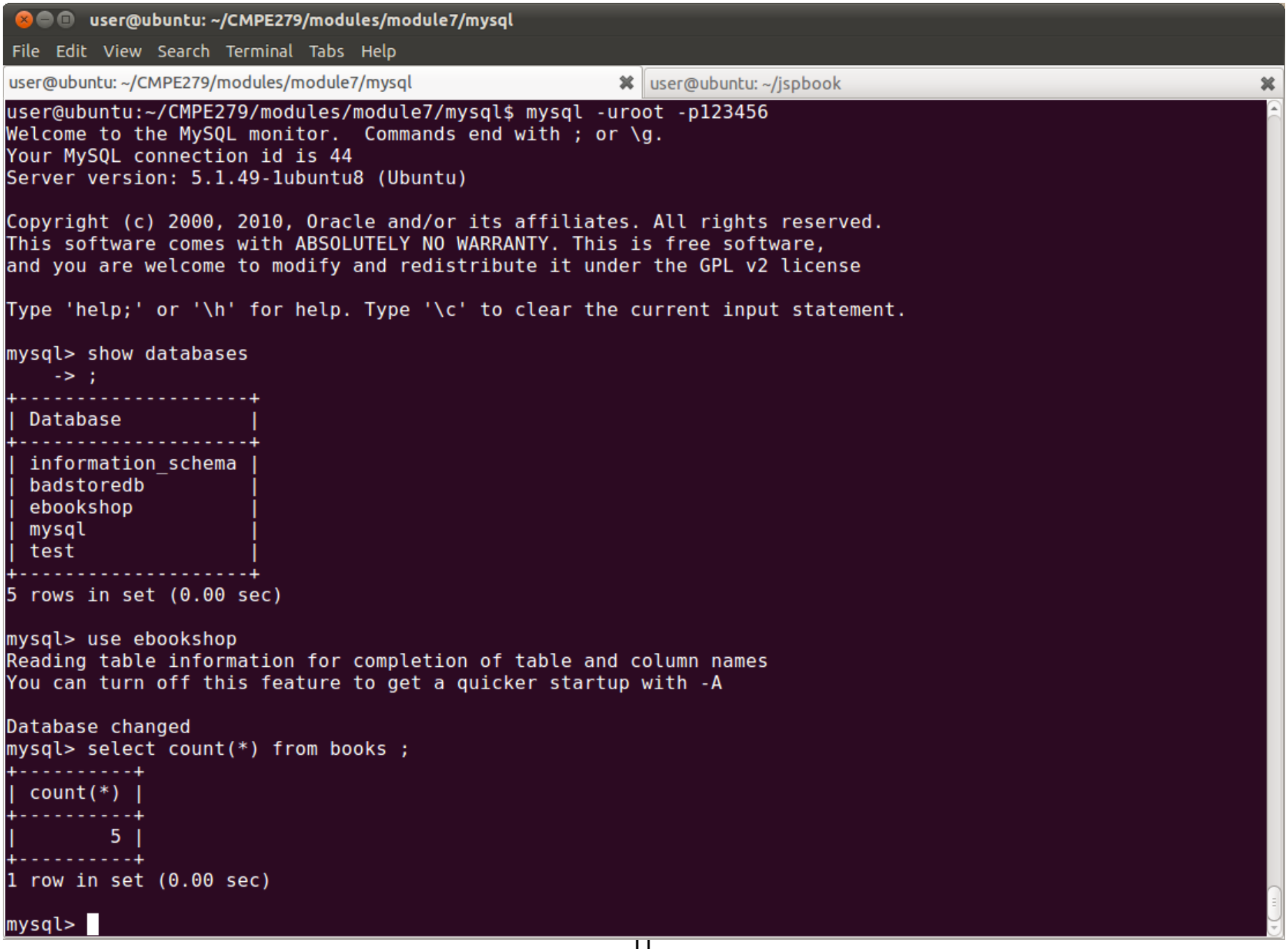
MySQL JDBC

JDBC Driver Setup and Test

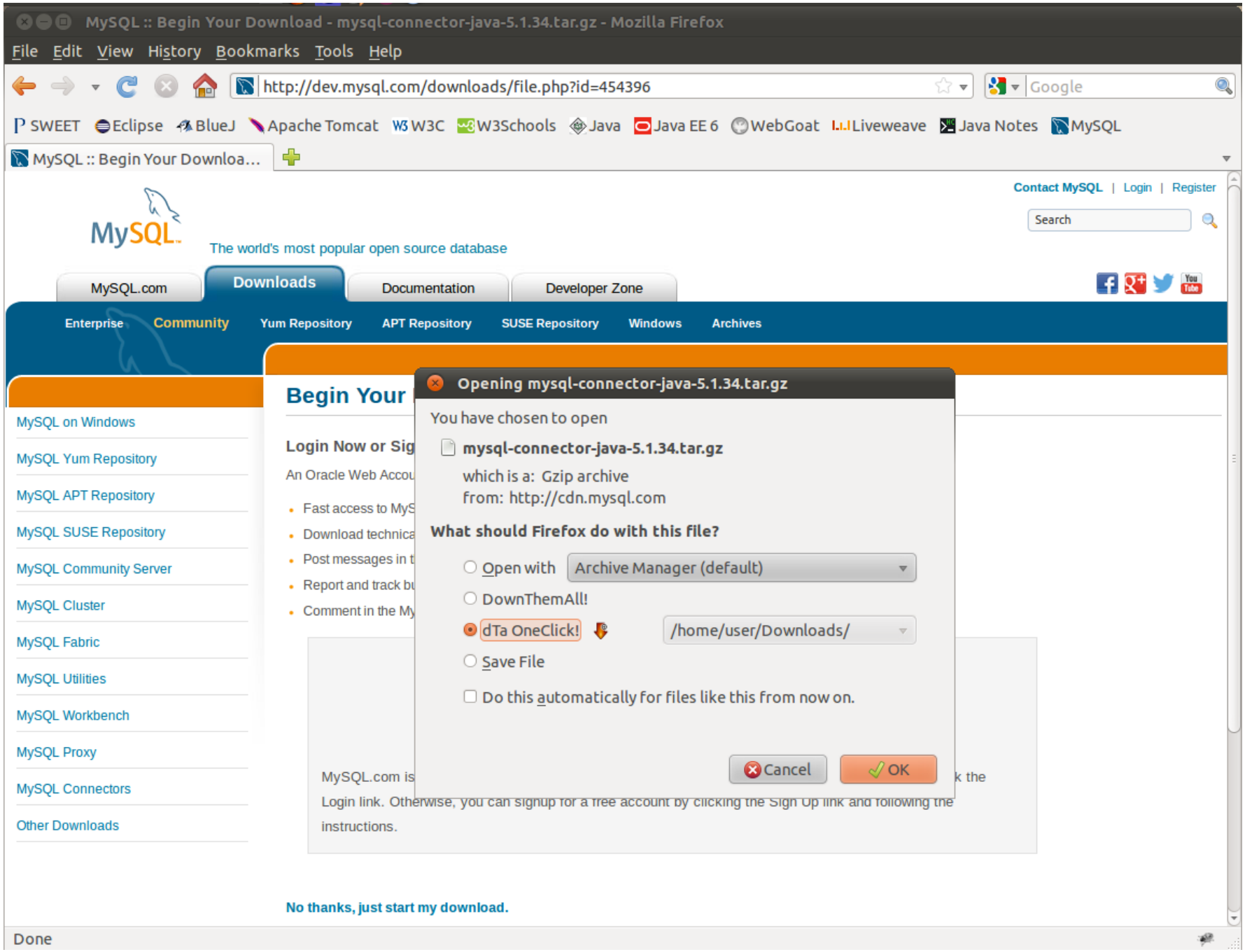
Setting Up Database: ebookshop
(see module7/mysql)



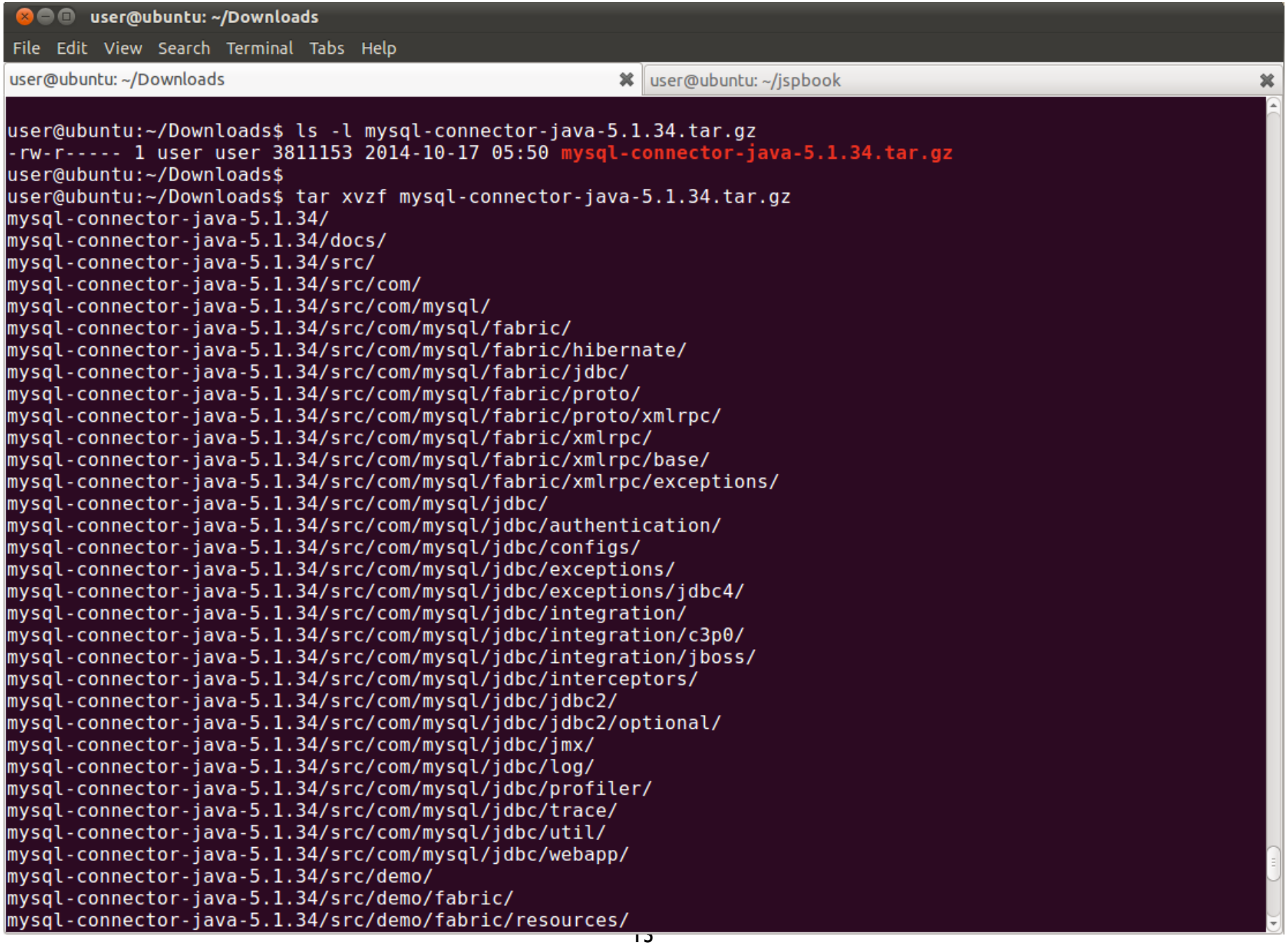
Listing of Database Schemas in MySQL



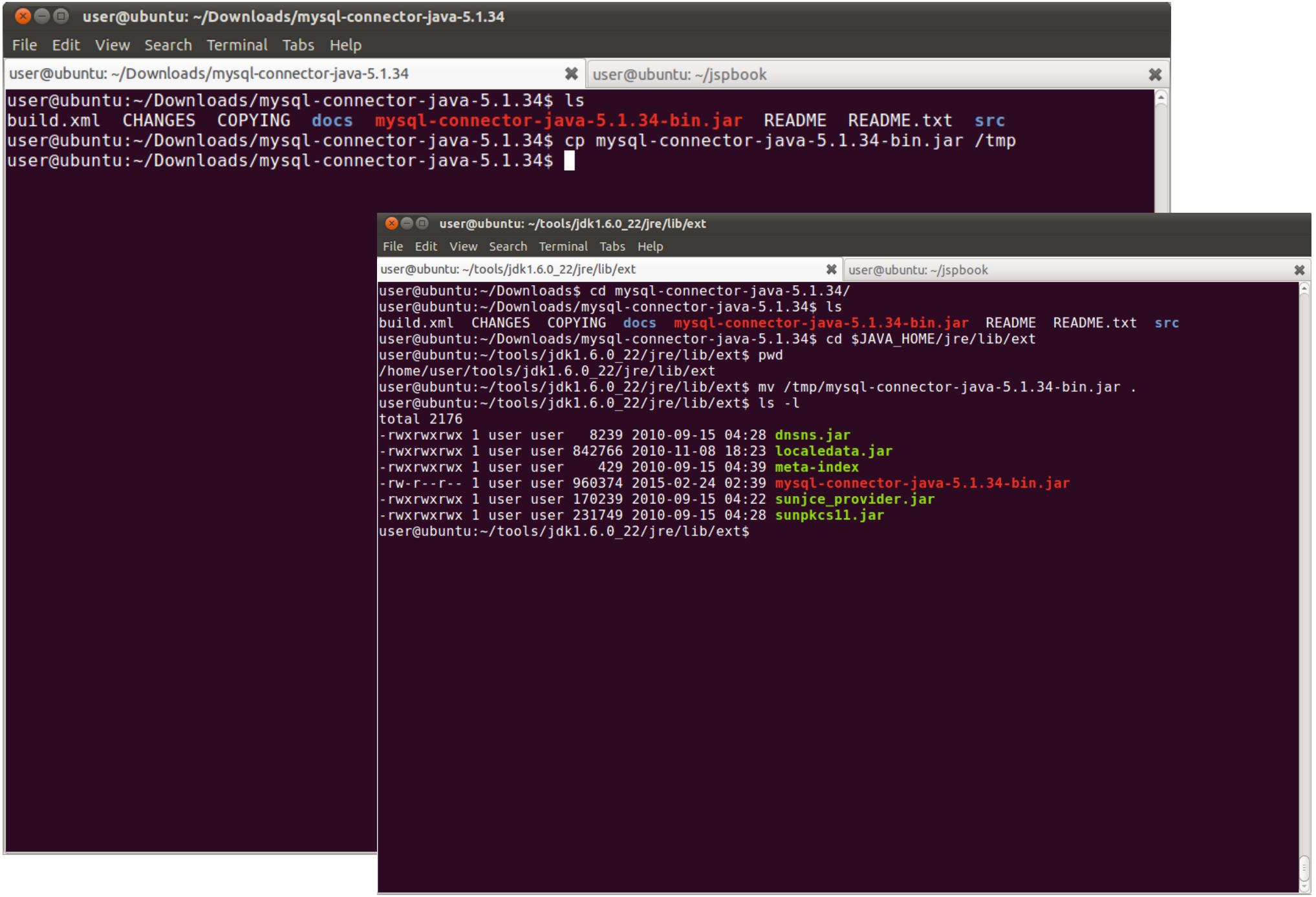
Download JDBC Driver for MySQL



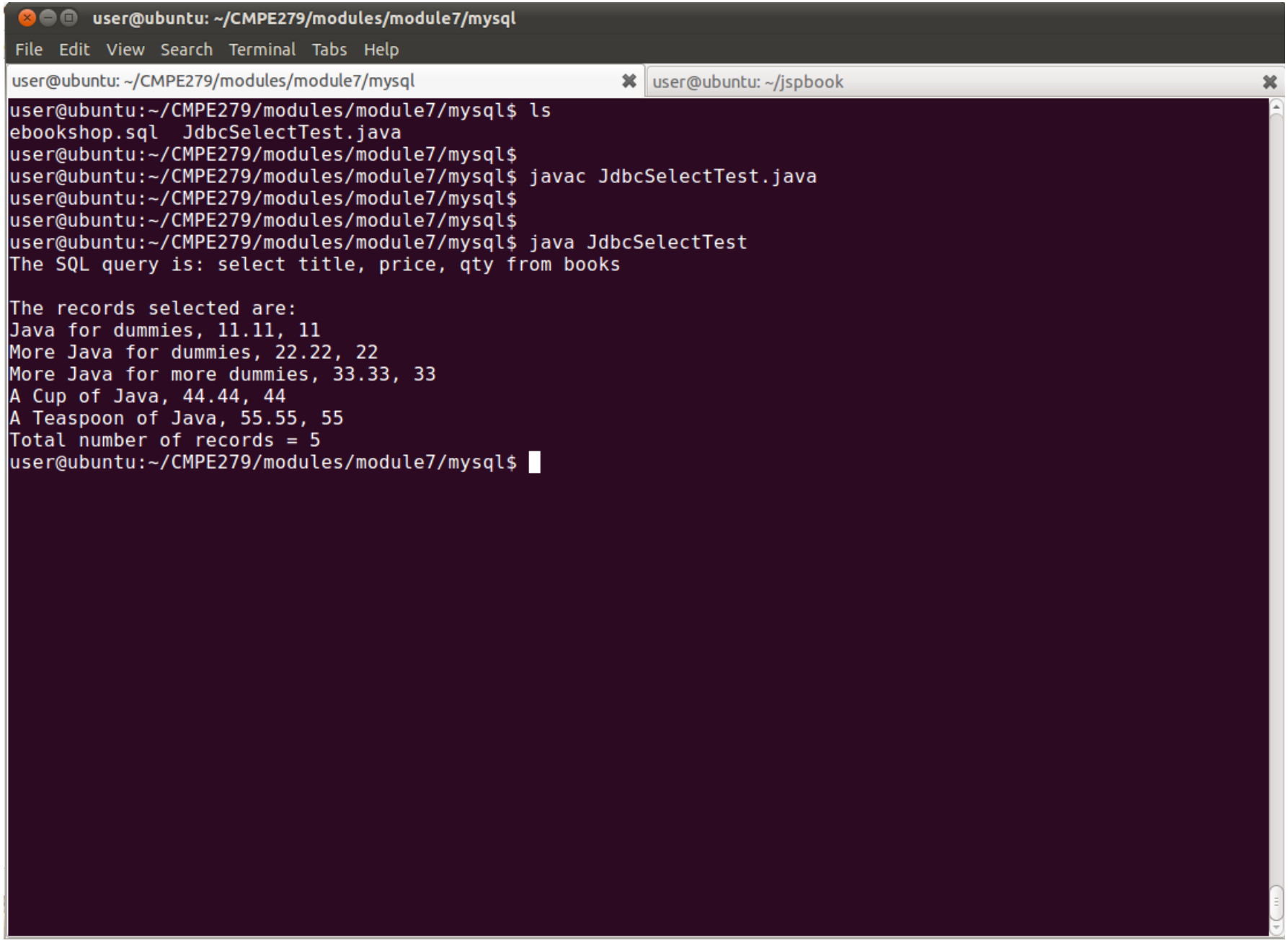
Extract Downloaded Tarball



Copy JDBC JAR to $JAVA_HOME/jre/lib/ext



Test JDBC Connection



JdbcSelectTest.java (~/CMPE279/modules/module7/mysql) - gedit

File Edit View Search Tools Documents Help

Open Save Undo

JdbcSelectTest.java

```
import java.sql.*;             // Use classes in java.sql package

// JDK 6 and above
public class JdbcSelectTest {   // Save as JdbcSelectTest.java
   public static void main(String[] args) {
      Connection conn = null;
      Statement  stmt = null;
      try {
         // Step 1: Allocate a database "Connection" object
         conn = DriverManager.getConnection(
               "jdbc:mysql://localhost:3306/ebookshop", "root", "123456"); // MySQL

         // Step 2: Allocate a "Statement" object in the Connection
         stmt = conn.createStatement();

         // Step 3: Execute a SQL SELECT query, the query result
         //  is returned in a "ResultSet" object.
         String strSelect = "select title, price, qty from books";
         System.out.println("The SQL query is: " + strSelect); // Echo For debugging
         System.out.println();

         ResultSet rset = stmt.executeQuery(strSelect);

         // Step 4: Process the ResultSet by scrolling the cursor forward via next().
         //  For each row, retrieve the contents of the cells with getXxx(columnName).
         System.out.println("The records selected are:");
         int rowCount = 0;
         while(rset.next()) {   // Move the cursor to the next row
            String title = rset.getString("title");
            double price = rset.getDouble("price");
            int    qty   = rset.getInt("qty");
            System.out.println(title + ", " + price + ", " + qty);
            ++rowCount;
         }
         System.out.println("Total number of records = " + rowCount);

      } catch(SQLException ex) {
         ex.printStackTrace();
      } finally {
         // Step 5: Always free resources
         try {
```

Java Tab Width: 8 Ln 1, Col 1 INS

Servlets & JSP

Workspace Setup

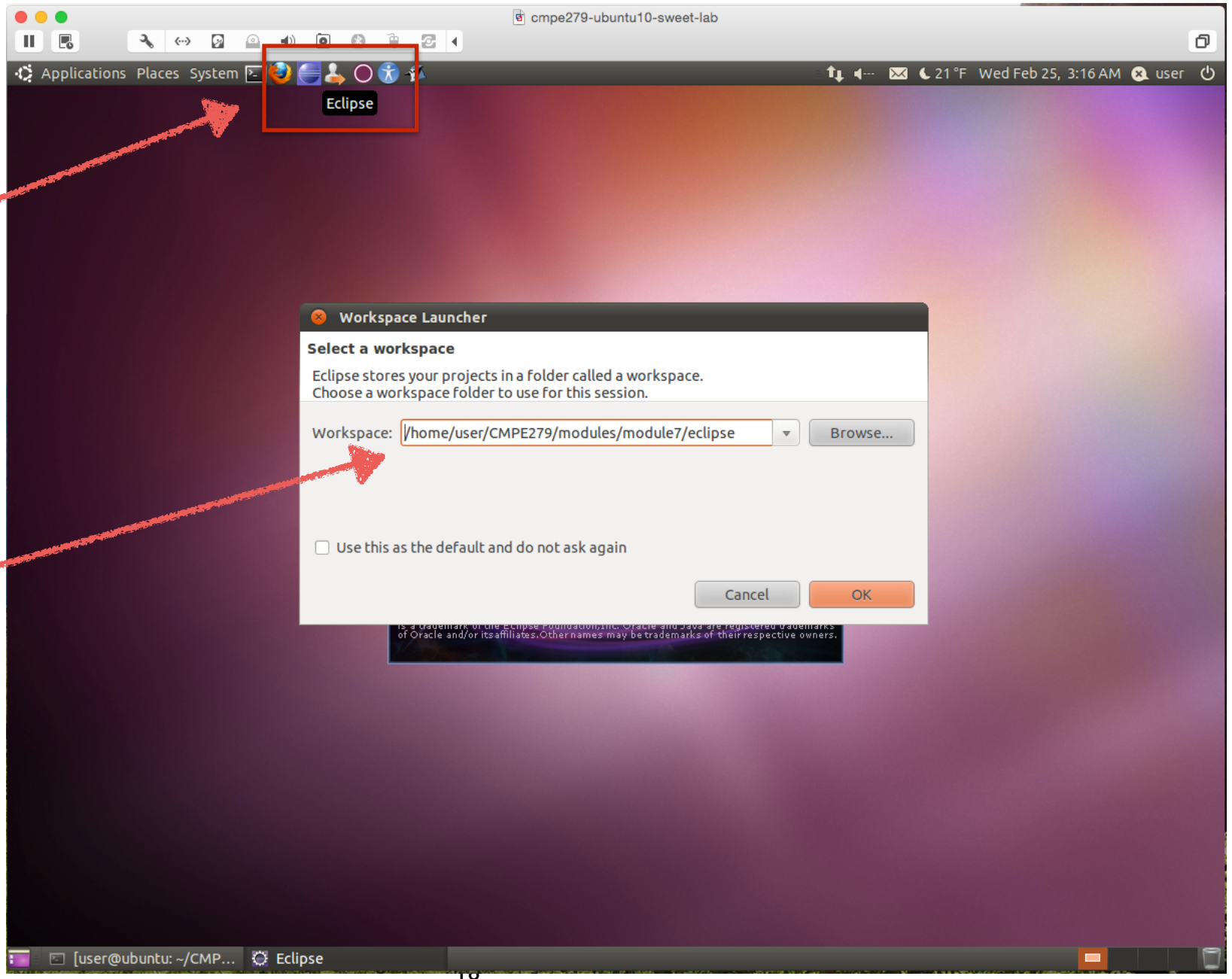
https://www.safaribooksonline.com/library/view/servlet-jsp-and/9781771970020/

Setup Eclipse Workspaces
(in modules #7 & #8)

Eclipse

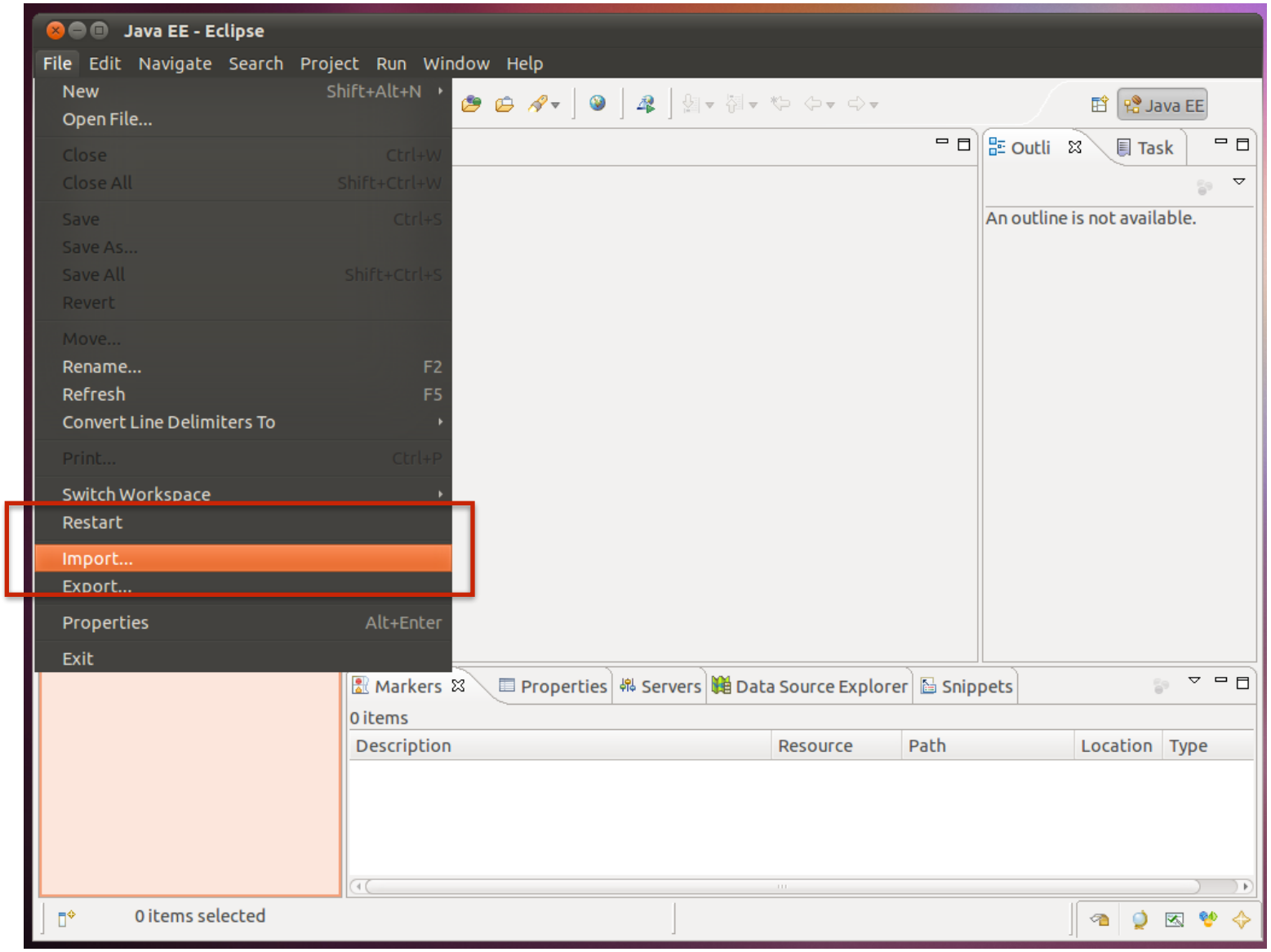
Launch

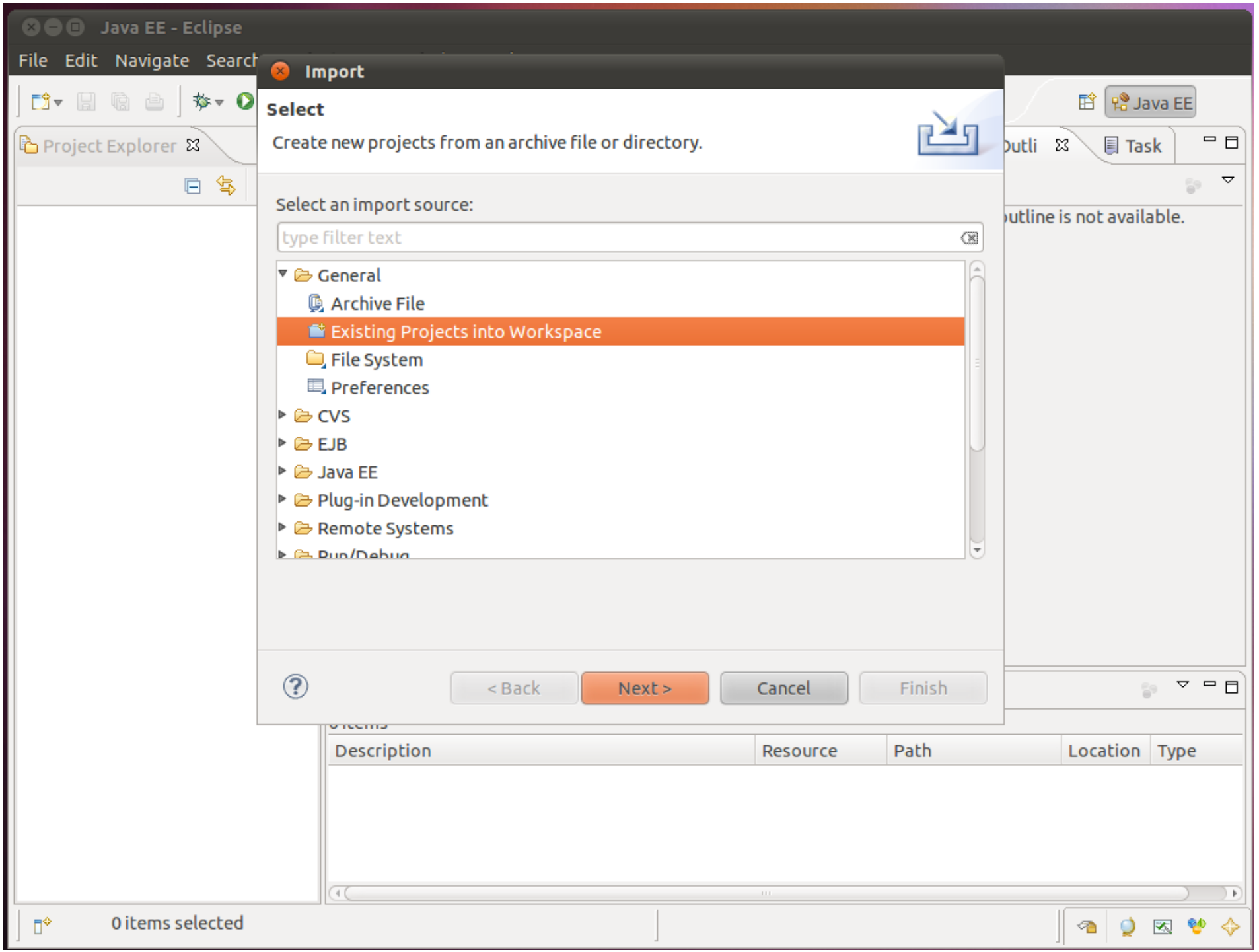
Select Workspace

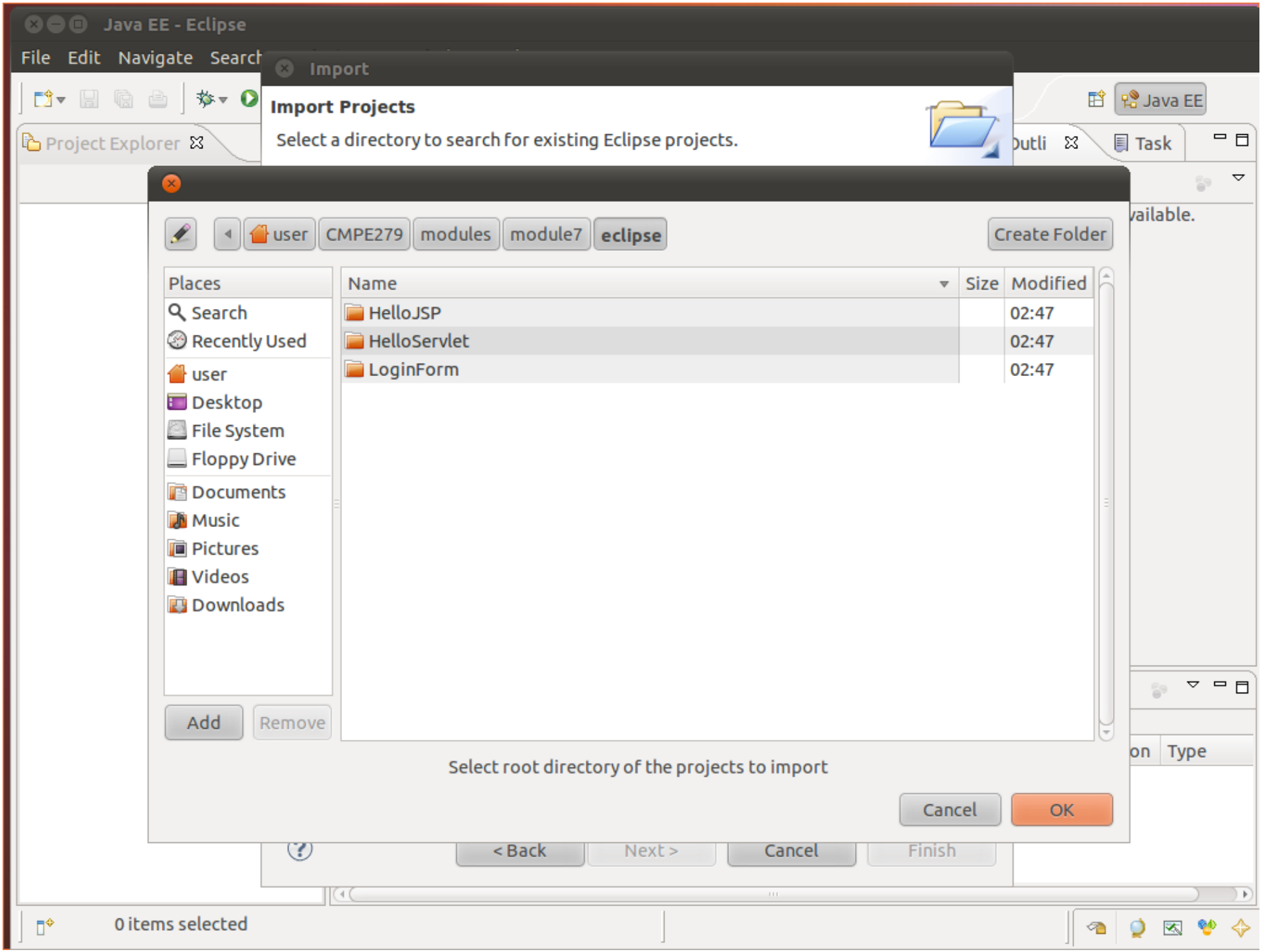


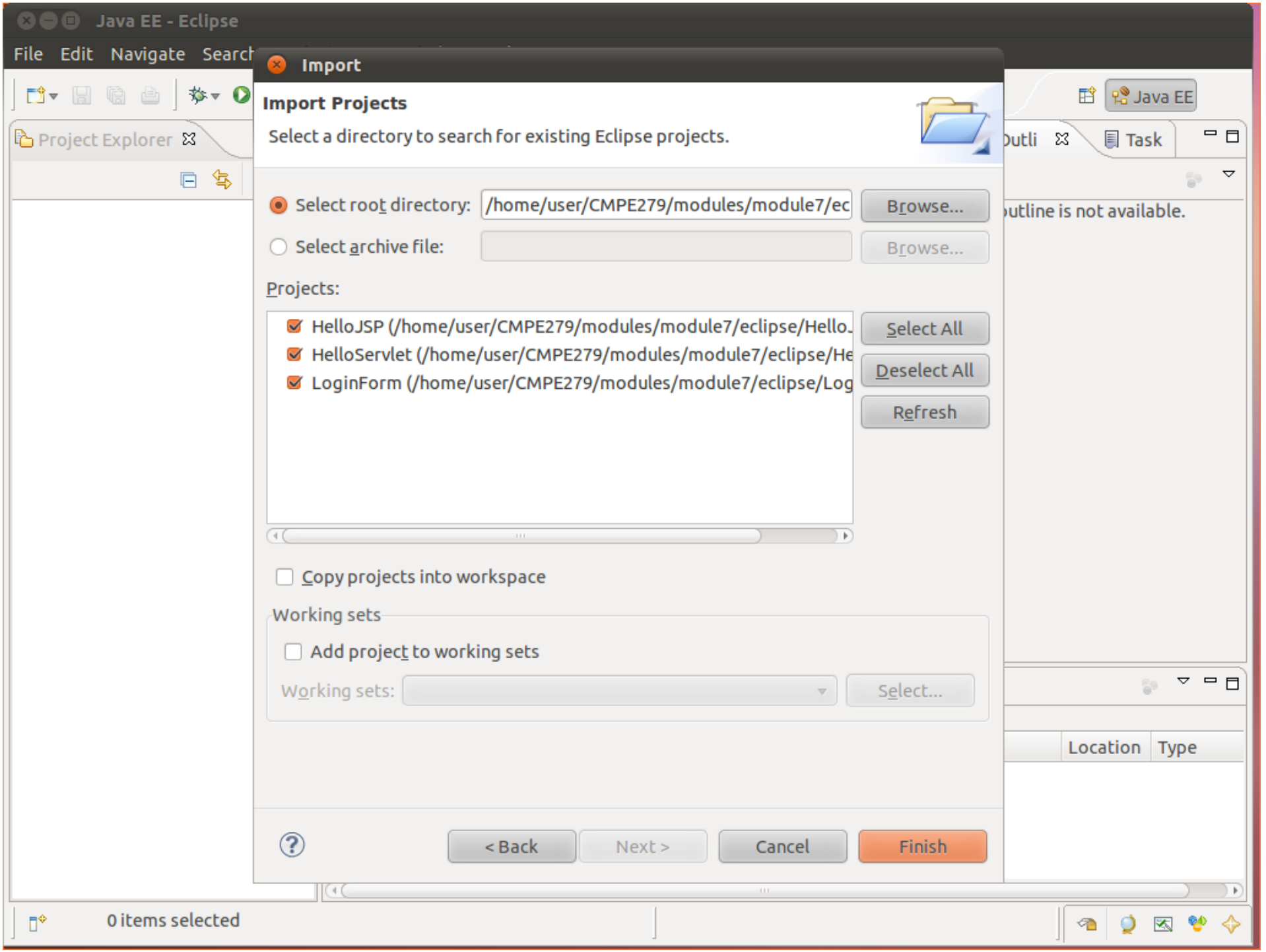
Eclipse

Import
Projects




Java EE - Eclipse
File Edit Navigate Search
Import
Select
Create new projects from an archive file or directory.
Select an import source:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
CVS
EJB
Java EE
Plug-in Development
Remote Systems
< Back
Next >
Cancel
Finish
Project Explorer
Java EE
Task
Description
Resource
Path
Location
Type
0 items selected

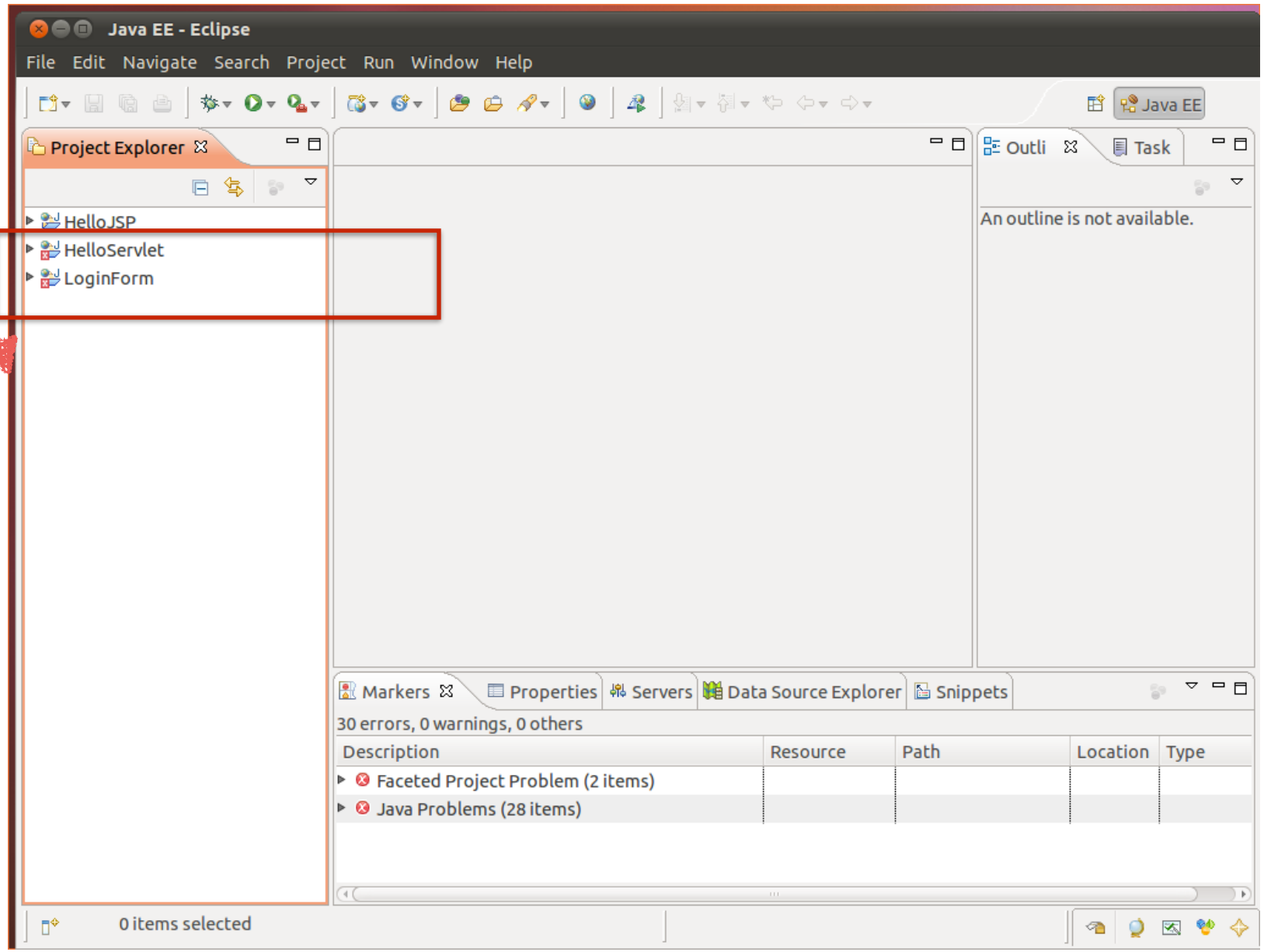

Java EE - Eclipse
File Edit Navigate Search
Import
Import Projects
Select a directory to search for existing Eclipse projects.
Java EE
Project Explorer
Task
user CMPE279 modules module7 eclipse
Create Folder
Places
Search
Recently Used
user
Desktop
File System
Floppy Drive
Documents
Music
Pictures
Videos
Downloads
Name
Size
Modified
HelloJSP 02:47
HelloServlet 02:47
LoginForm 02:47
Add
Remove
Select root directory of the projects to import
Cancel
OK
< Back
Next >
Cancel
Finish
0 items selected

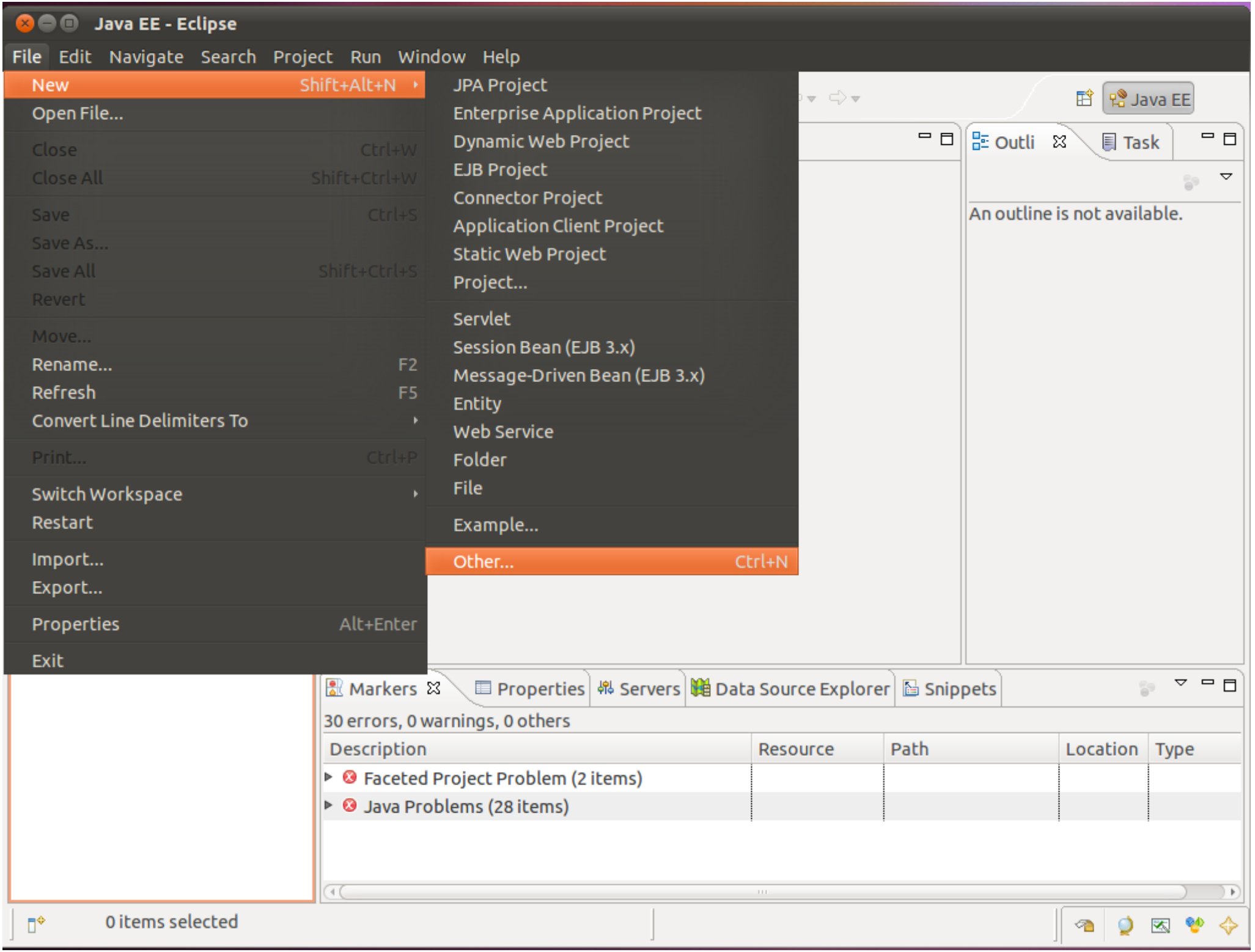

Java EE - Eclipse
File Edit Navigate Search
Import
Import Projects
Select a directory to search for existing Eclipse projects.
Select root directory:
/home/user/CMPE279/modules/module7/ec
Browse...
Select archive file:
Browse...
Projects:
HelloJSP (/home/user/CMPE279/modules/module7/eclipse/Hello
HelloServlet (/home/user/CMPE279/modules/module7/eclipse/He
LoginForm (/home/user/CMPE279/modules/module7/eclipse/Log
Select All
Deselect All
Refresh
Copy projects into workspace
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Cancel
Finish
Project Explorer
Java EE
Task
Outline is not available.
Location
Type
0 items selected

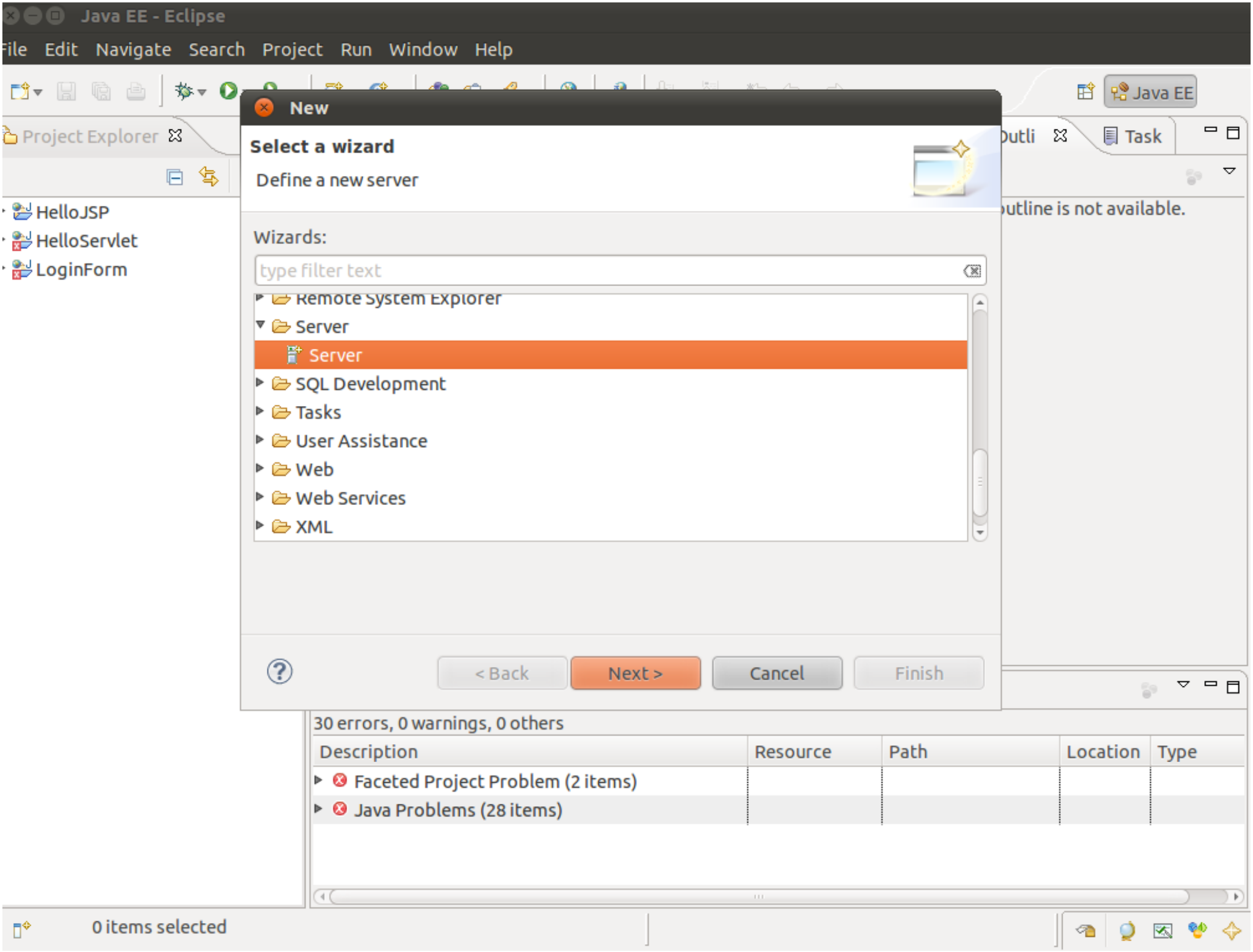
Errors!

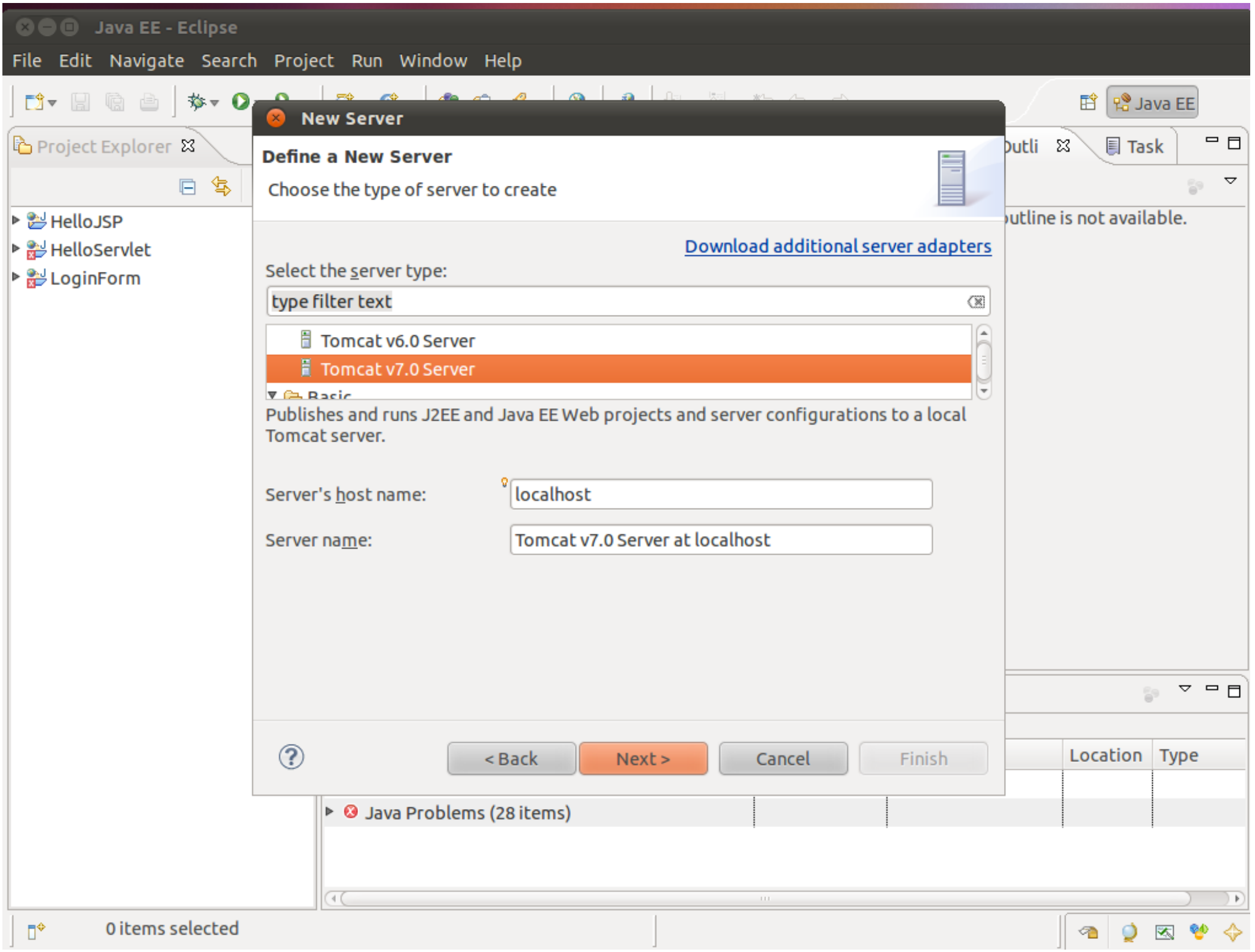
Errors
On Import

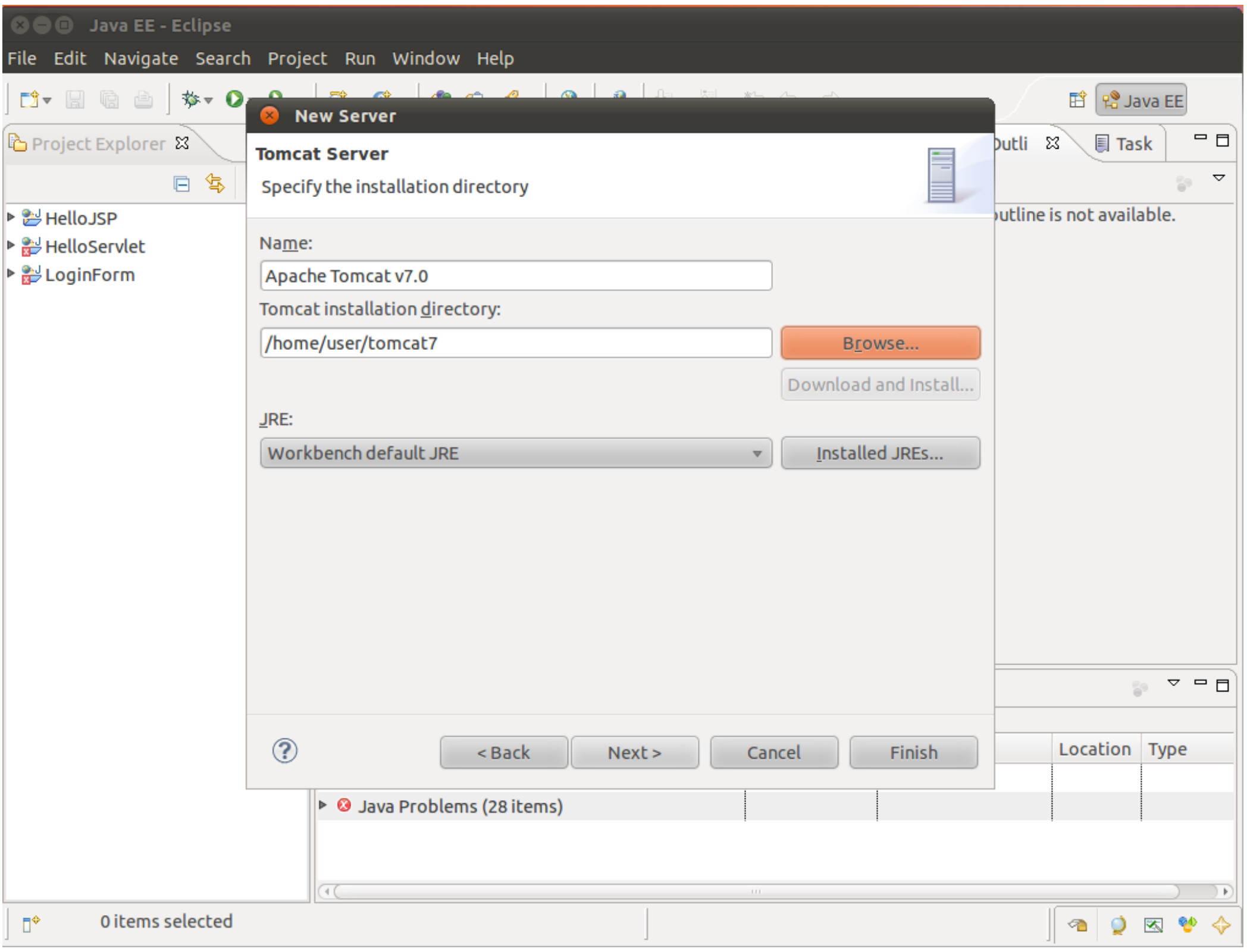
Need to
Setup
Tomcat
Servers

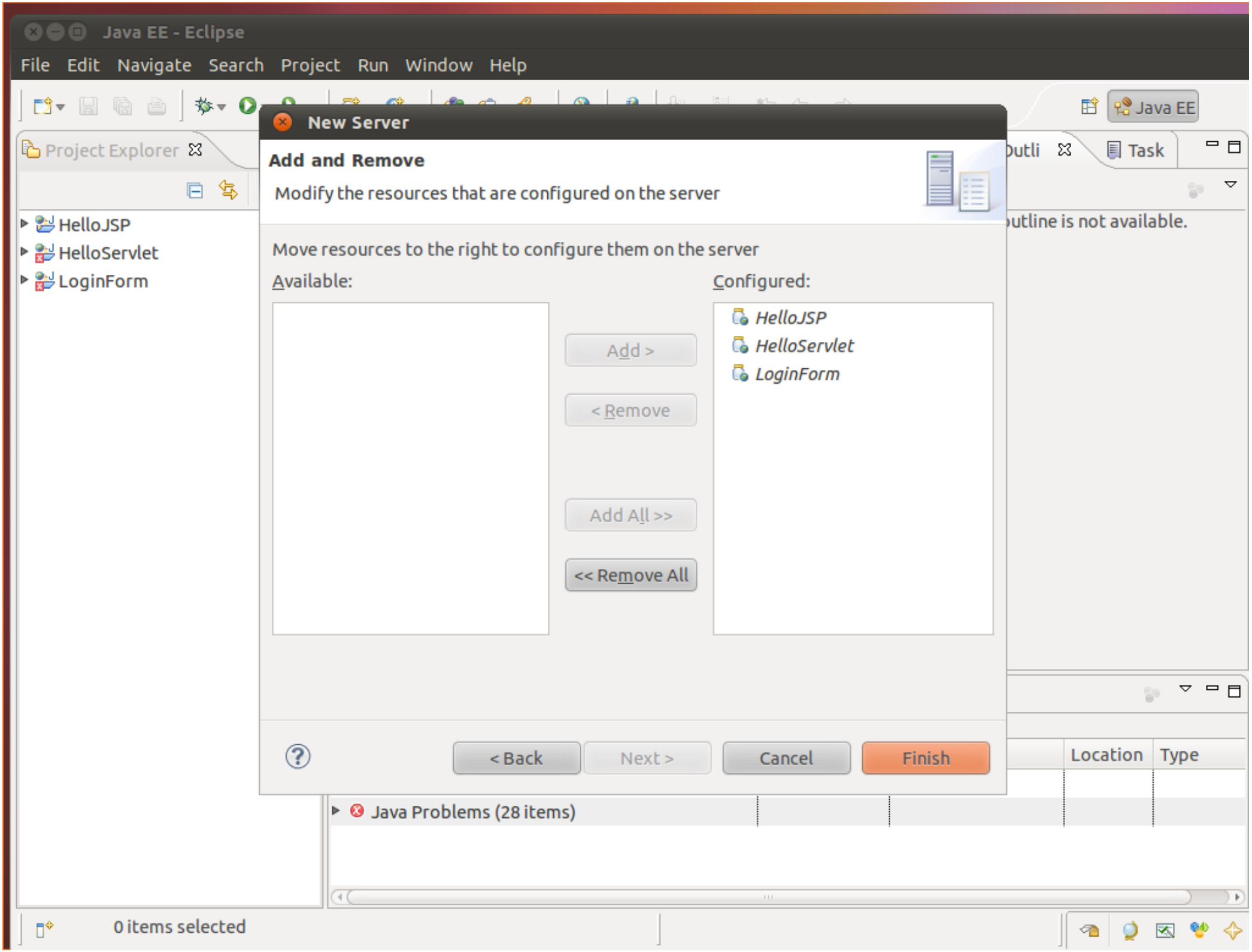


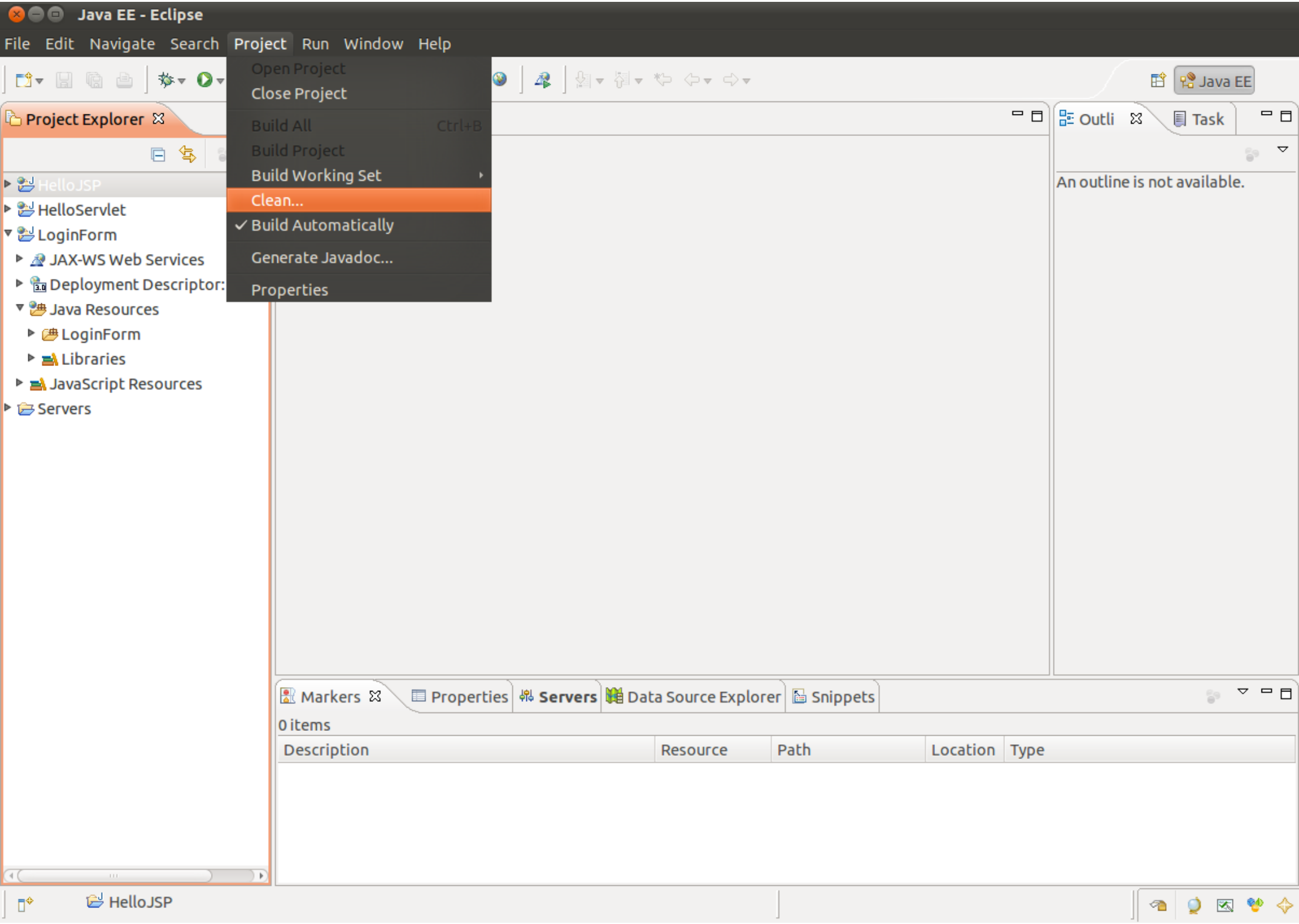

Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
New Shift+Alt+N
Open File...
Close Ctrl+W
Close All Shift+Ctrl+W
Save Ctrl+S
Save As...
Save All Shift+Ctrl+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
Exit
JPA Project
Enterprise Application Project
Dynamic Web Project
EJB Project
Connector Project
Application Client Project
Static Web Project
Project...
Servlet
Session Bean (EJB 3.x)
Message-Driven Bean (EJB 3.x)
Entity
Web Service
Folder
File
Example...
Other... Ctrl+N
Java EE
Outli
Task
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
30 errors, 0 warnings, 0 others
Description
Resource
Path
Location
Type
Faceted Project Problem (2 items)
Java Problems (28 items)
0 items selected

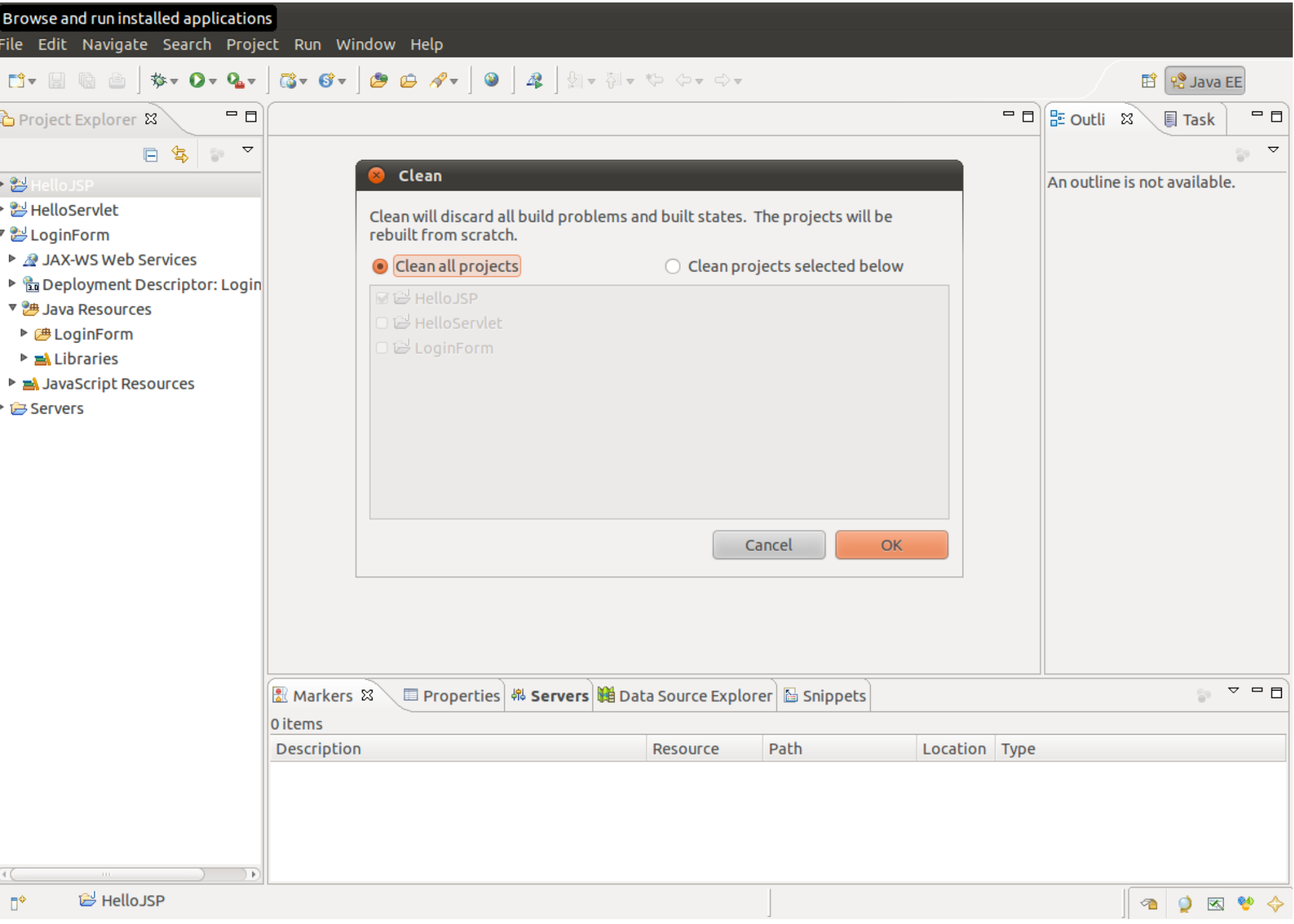

Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
HelloJSP
HelloServlet
LoginForm
Outli
Task
Outline is not available.
New
Select a wizard
Define a new server
Wizards:
type filter text
Remote System Explorer
Server
Server
SQL Development
Tasks
User Assistance
Web
Web Services
XML
< Back
Next >
Cancel
Finish
30 errors, 0 warnings, 0 others
Description
Resource
Path
Location
Type
Faceted Project Problem (2 items)
Java Problems (28 items)
0 items selected


Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
HelloJSP
HelloServlet
LoginForm
Task
utline is not available.
New Server
Define a New Server
Choose the type of server to create
Download additional server adapters
Select the server type:
type filter text
Tomcat v6.0 Server
Tomcat v7.0 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 v7.0 Server at localhost
< Back
Next >
Cancel
Finish
Location
Type
Java Problems (28 items)
0 items selected

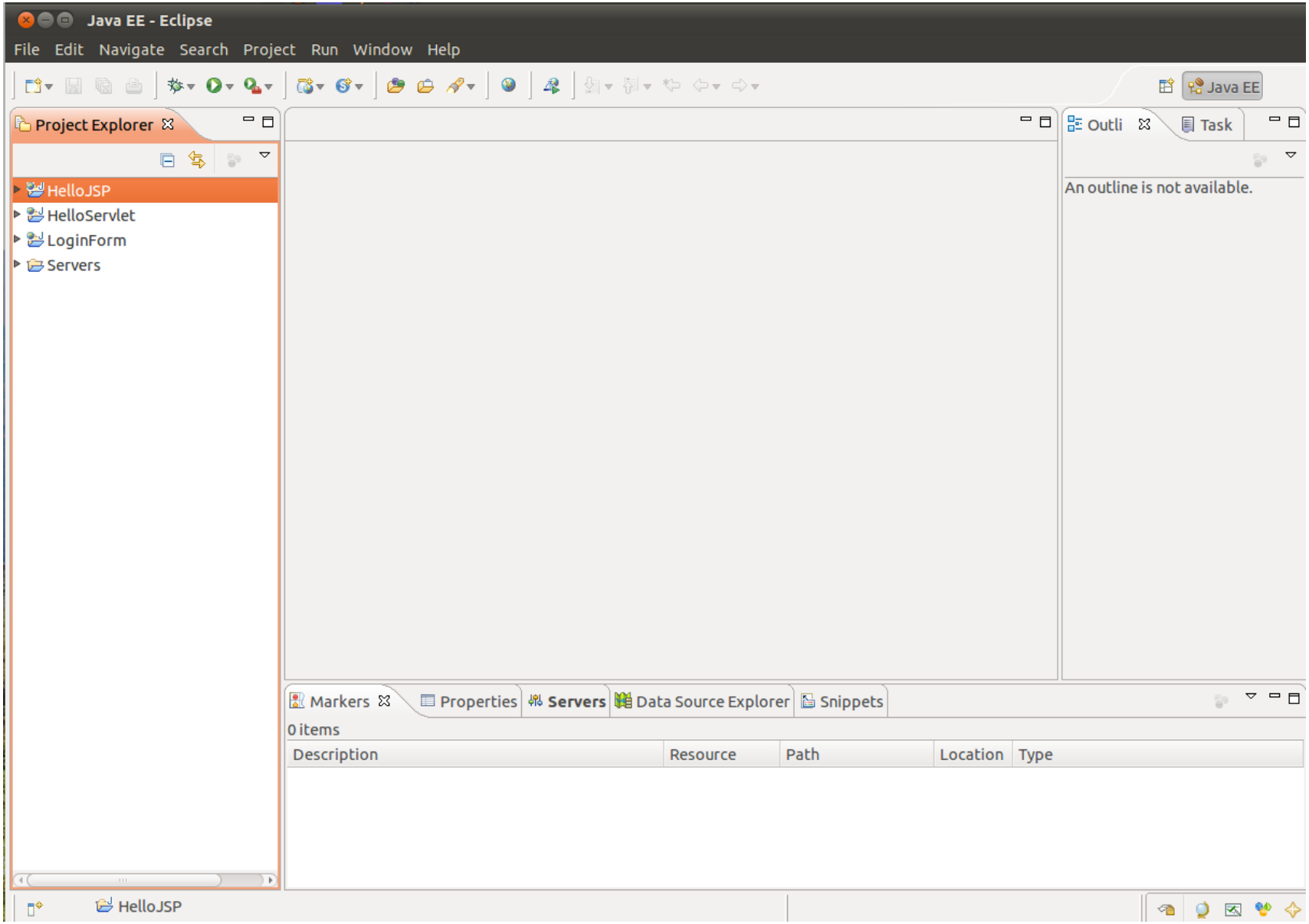

Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
HelloJSP
HelloServlet
LoginForm
Task
utline is not available.
New Server
Tomcat Server
Specify the installation directory
Name:
Apache Tomcat v7.0
Tomcat installation directory:
/home/user/tomcat7
Browse...
Download and Install...
JRE:
Workbench default JRE
Installed JREs...
< Back
Next >
Cancel
Finish
Location
Type
Java Problems (28 items)
0 items selected


Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
HelloJSP
HelloServlet
LoginForm
New Server
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
HelloJSP
HelloServlet
LoginForm
Add >
< Remove
Add All >>
<< Remove All
< Back
Next >
Cancel
Finish
Task
utline is not available.
Location
Type
Java Problems (28 items)
0 items selected


Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Open Project
Close Project
Build All Ctrl+B
Build Project
Build Working Set
Clean...
Build Automatically
Generate Javadoc...
Properties
Java EE
Project Explorer
HelloJSP
HelloServlet
LoginForm
JAX-WS Web Services
Deployment Descriptor:
Java Resources
LoginForm
Libraries
JavaScript Resources
Servers
Outli
Task
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
0 items
Description
Resource
Path
Location
Type
HelloJSP


Browse and run installed applications
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
HelloJSP
HelloServlet
LoginForm
JAX-WS Web Services
Deployment Descriptor: Login
Java Resources
LoginForm
Libraries
JavaScript Resources
Servers
Clean
Clean will discard all build problems and built states. The projects will be rebuilt from scratch.
Clean all projects
Clean projects selected below
HelloJSP
HelloServlet
LoginForm
Cancel
OK
Outli
Task
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
0 items
Description
Resource
Path
Location
Type
HelloJSP

Review & Execute Projects…



Module8

Setup workspace in ~/CMPE279/modules/module8/eclipse

Servlets & JSP

Examples in Module7

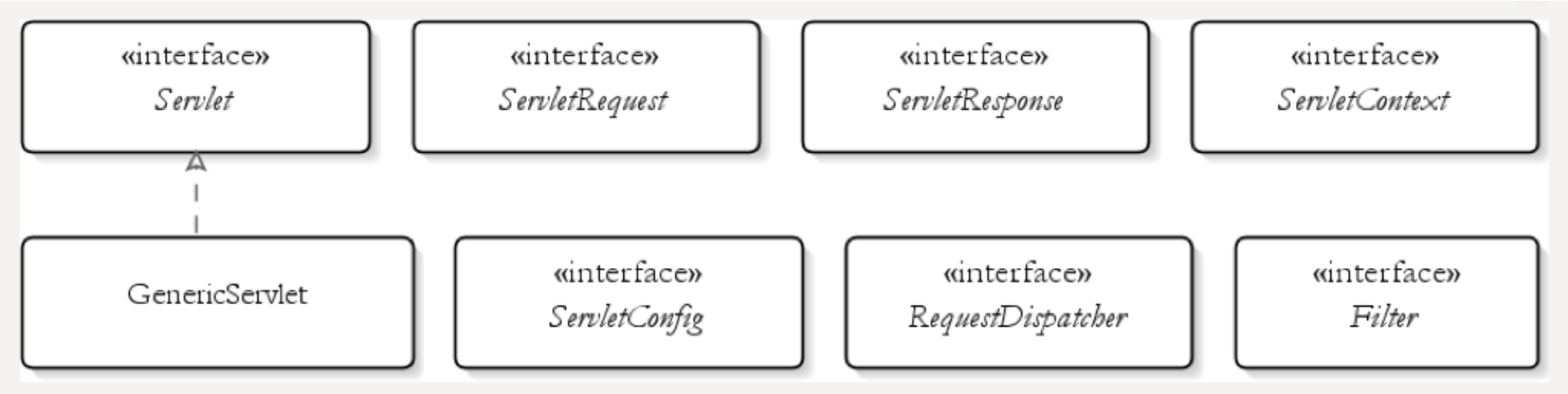
https://www.safaribooksonline.com/library/view/servlet-jsp-and/9781771970020/

Key Servlet Interfaces

Servlet API Overview

The Servlet API comes in four Java packages. The packages are as follows.

- **javax.servlet**. Contains classes and interfaces that define the contract between a servlet and a servlet container.
- **javax.servlet.http**. Contains classes and interfaces that define the contract between an HTTP servlet and a servlet container.
- **javax.servlet.annotation**. Contains annotations to annotate servlets, filters, and listeners. It also specifies metadata for annotated components.
- **javax.servlet.descriptor**. Contains types that provide programmatic access to a web application's configuration information.

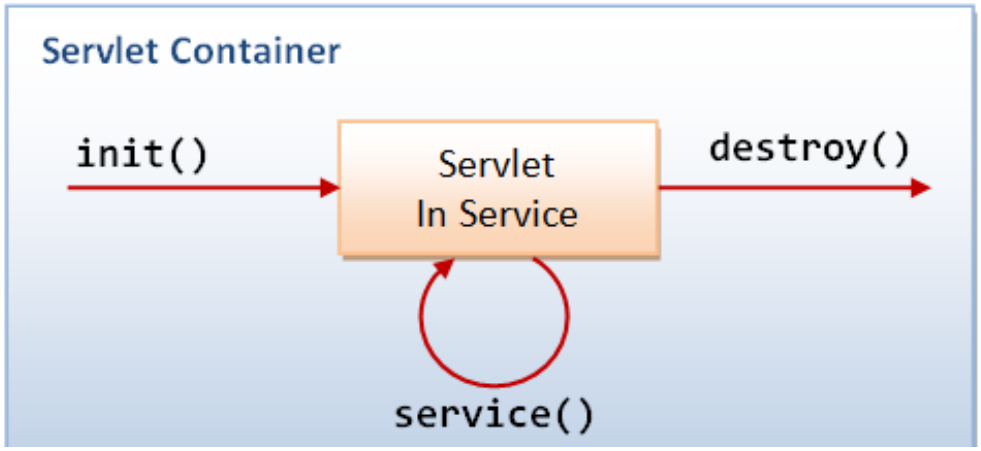


Servlet Life Cycle

Servlet

The **Servlet** interface defines these five methods.

```
void init(ServletConfig config) throws ServletException
void service(ServletRequest request, ServletResponse response)
        throws ServletException, java.io.IOException
void destroy()
java.lang.String getServletInfo()
ServletConfig getServletConfig()
```



Java EE - HelloServlet/src/GenericServletDemoServlet.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Java EE

Project Explorer

- HelloJSP
- HelloServlet
  - JAX-WS Web Services
  - Deployment Descriptor: Hello
  - Java Resources
    - src
      - (default package)
        - GenericServletDemoServl
        - MyServlet.java
        - ServletConfigDemoServle
    - Libraries
  - JavaScript Resources
  - build
  - WebContent
- LoginForm
- Servers

GenericServletDemoServlet.java

```
import java.io.IOException;

@WebServlet(name = "GenericServletDemoServlet",
    urlPatterns = { "/generic" },
    initParams = {
        @WebInitParam(name="admin", value="Harry Taciak"),
        @WebInitParam(name="email", value="admin@example.com")
    }
)
public class GenericServletDemoServlet extends GenericServlet {

    private static final long serialVersionUID = 62500890L;

    @Override
    public void service(ServletRequest request,
            ServletResponse response)
            throws ServletException, IOException {
        ServletConfig servletConfig = getServletConfig();
        String admin = servletConfig.getInitParameter("admin");
        String email = servletConfig.getInitParameter("email");
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();
        writer.print("<html><head></head><body>" +
                "Admin:" + admin +
                "<br/>Email:" + email +
                "</body></html>");
    }
}
```

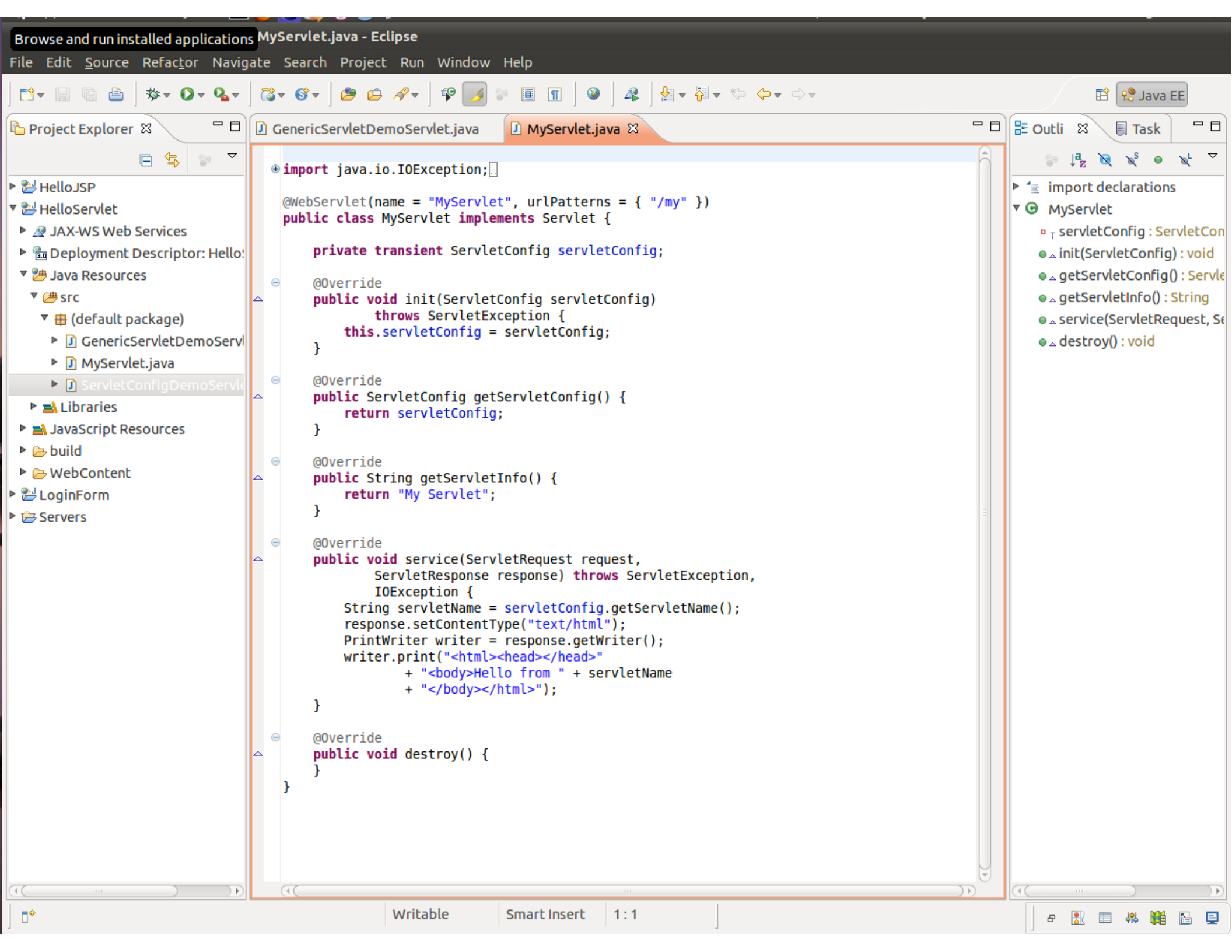
Outli Task

- import declarations
- GenericServletDemoServlet
  - serialVersionUID : long
  - service(ServletRequest, Se

Markers Properties Servers Data Source Explorer Snippets Console

Tomcat v7.0 Server at localhost [Stopped, Republish]

GenericServletDemoServlet.java - HelloServlet/src


Browse and run installed applications
MyServlet.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
HelloJSP
HelloServlet
JAX-WS Web Services
Deployment Descriptor: Hello
Java Resources
src
(default package)
GenericServletDemoServl
MyServlet.java
ServletConfigDemoServle
Libraries
JavaScript Resources
build
WebContent
LoginForm
Servers
GenericServletDemoServlet.java
MyServlet.java
import java.io.IOException;
@WebServlet(name = "MyServlet", urlPatterns = { "/my" })
public class MyServlet implements Servlet {
private transient ServletConfig servletConfig;
@Override
public void init(ServletConfig servletConfig)
throws ServletException {
this.servletConfig = servletConfig;
}
@Override
public ServletConfig getServletConfig() {
return servletConfig;
}
@Override
public String getServletInfo() {
return "My Servlet";
}
@Override
public void service(ServletRequest request,
ServletResponse response) throws ServletException,
IOException {
String servletName = servletConfig.getServletName();
response.setContentType("text/html");
PrintWriter writer = response.getWriter();
writer.print("<html><head></head>"
+ "<body>Hello from " + servletName
+ "</body></html>");
}
@Override
public void destroy() {
}
}
Outli
Task
import declarations
MyServlet
servletConfig : ServletCon
init(ServletConfig) : void
getServletConfig() : Servle
getServletInfo() : String
service(ServletRequest, Se
destroy() : void
Writable
Smart Insert
1 : 1

Java EE - HelloServlet/src/ServletConfigDemoServlet.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Java EE

Project Explorer

- HelloJSP
- HelloServlet
  - JAX-WS Web Services
  - Deployment Descriptor: Hello
  - Java Resources
    - src
      - (default package)
        - GenericServletDemoServl
        - MyServlet.java
        - ServletConfigDemoServle
    - Libraries
  - JavaScript Resources
  - build
  - WebContent
- LoginForm
- Servers

GenericServletDemoServlet.java | MyServlet.java | ServletConfigDemoServlet.java

```
import java.io.IOException;

@WebServlet(name = "ServletConfigDemoServlet",
    urlPatterns = { "/servletConfigDemo" },
    initParams = {
        @WebInitParam(name="admin", value="Harry Taciak"),
        @WebInitParam(name="email", value="admin@example.com")
    }
)
public class ServletConfigDemoServlet implements Servlet {
    private transient ServletConfig servletConfig;

    @Override
    public ServletConfig getServletConfig() {
        return servletConfig;
    }

    @Override
    public void init(ServletConfig servletConfig)
            throws ServletException {
        this.servletConfig = servletConfig;
    }

    @Override
    public void service(ServletRequest request,
            ServletResponse response)
            throws ServletException, IOException {
        ServletConfig servletConfig = getServletConfig();
        String admin = servletConfig.getInitParameter("admin");
        String email = servletConfig.getInitParameter("email");
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();
        writer.print("<html><head></head><body>" +
                "Admin:" + admin +
                "<br/>Email:" + email +
                "</body></html>");
    }

    @Override
    public String getServletInfo() {
        return "ServletConfig demo";
    }

    @Override
```

Outli | Task

- import declarations
- ServletConfigDemoServlet
  - servletConfig : ServletCon
  - getServletConfig() : Servle
  - init(ServletConfig) : void
  - service(ServletRequest, Se
  - getServletInfo() : String
  - destroy() : void

ServletConfigDemoServlet.java - HelloServlet/src

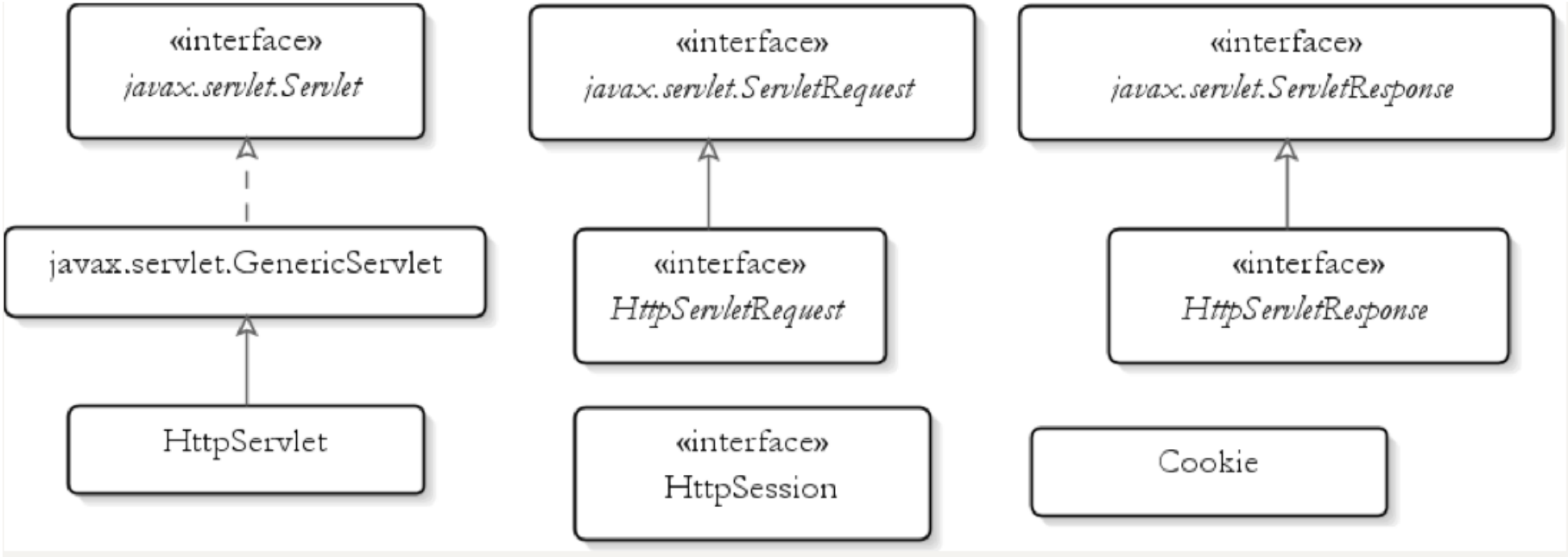
Servlets & JSP

Examples in Module8

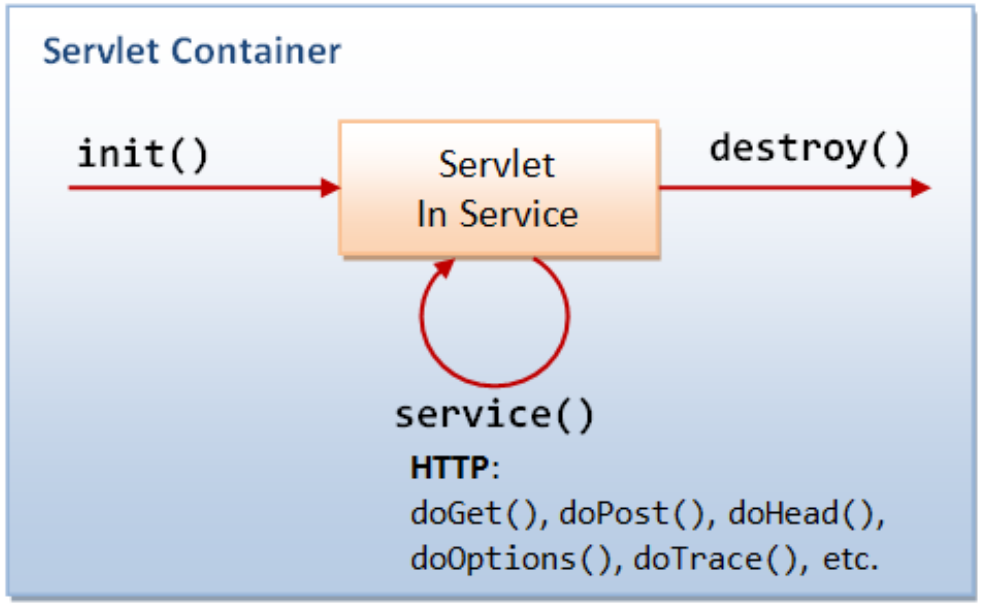
https://www.safaribooksonline.com/library/view/servlet-jsp-and/9781771970020/

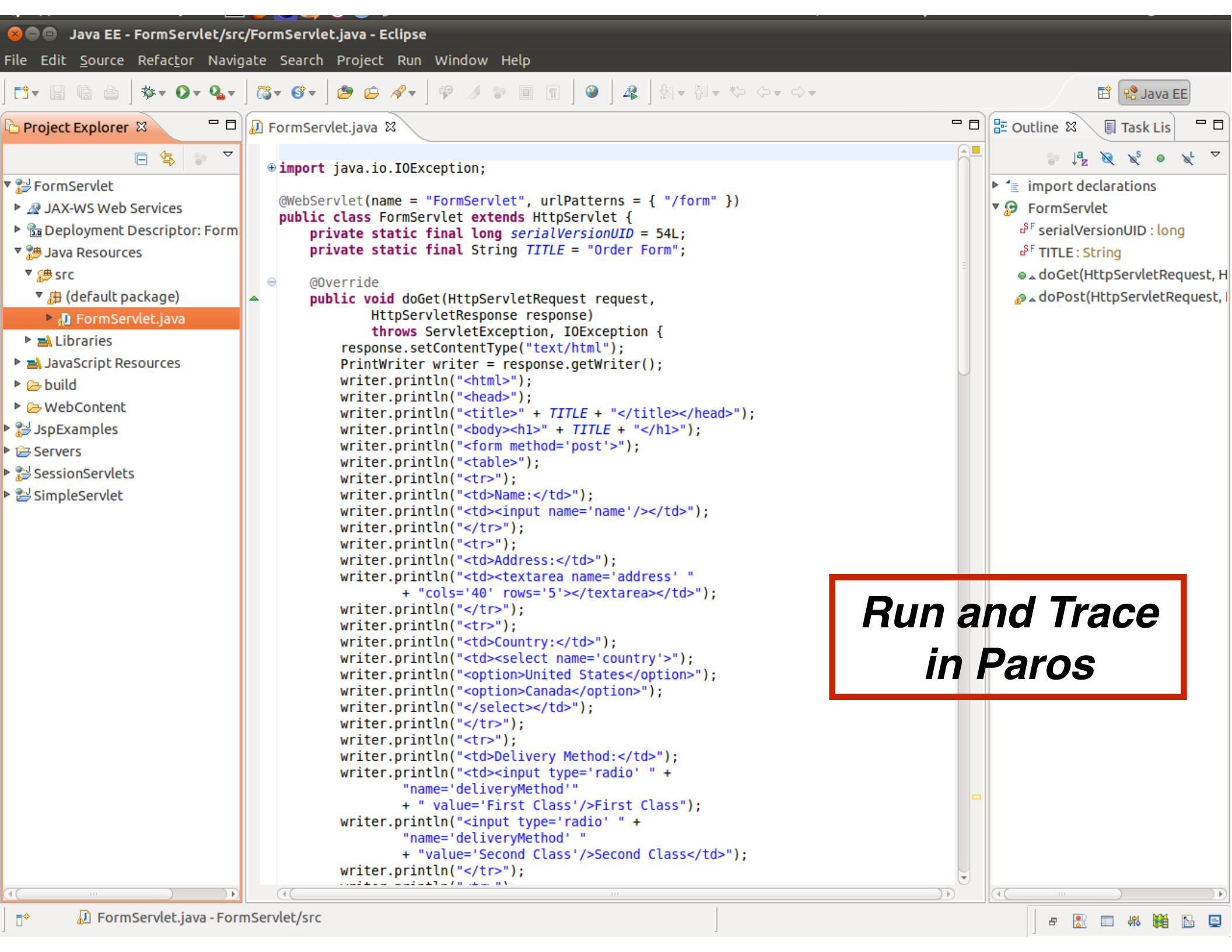
HTTP Servlets

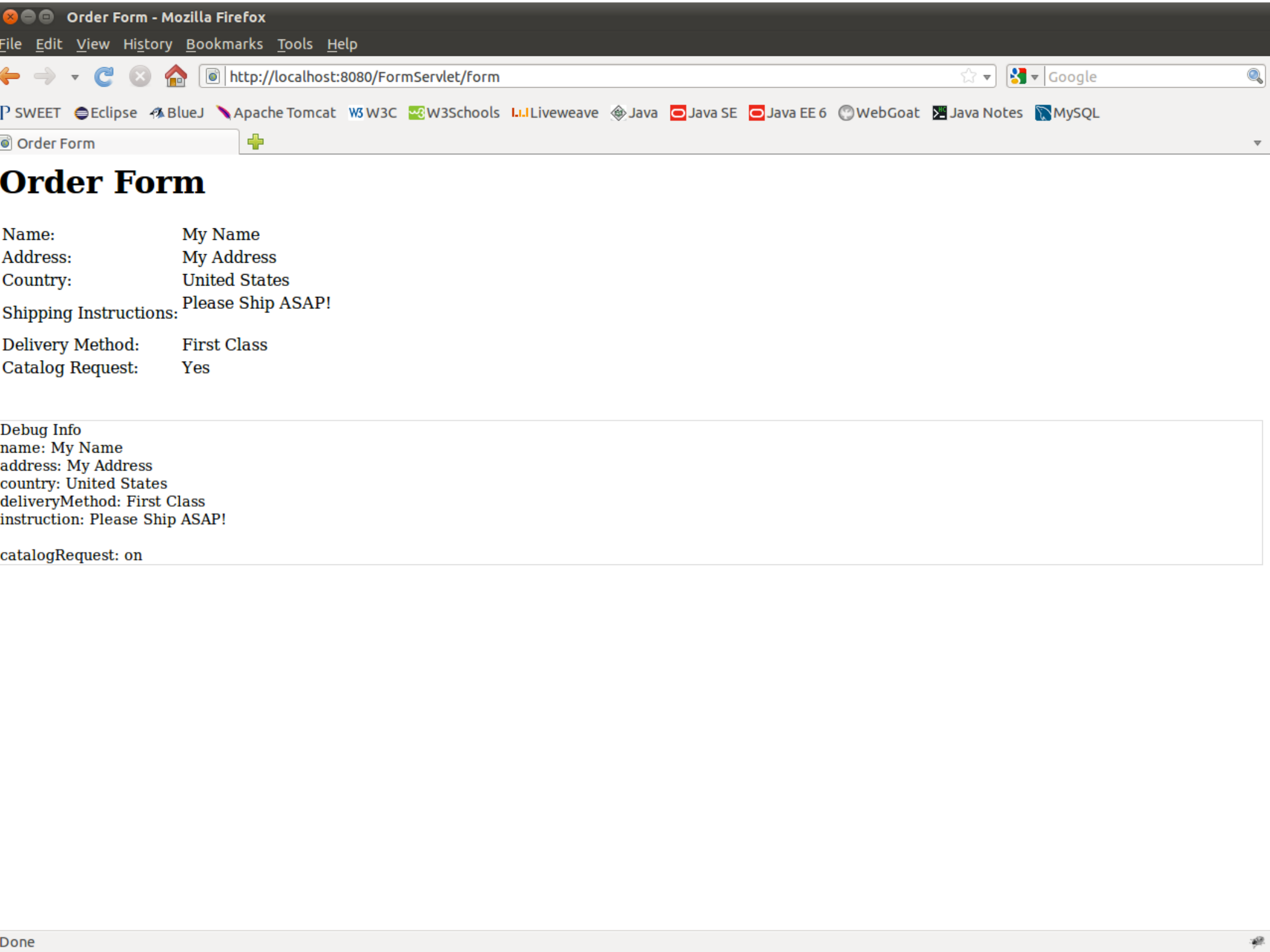
Most, if not all, servlet applications you write will work with HTTP. This means, you can make use of the features offered by HTTP. The **javax.servlet.http** package is the second package in the Servlet API that contains classes and interfaces for writing servlet applications. Many of the types in **javax.servlet.http** override those in **javax.servlet**.

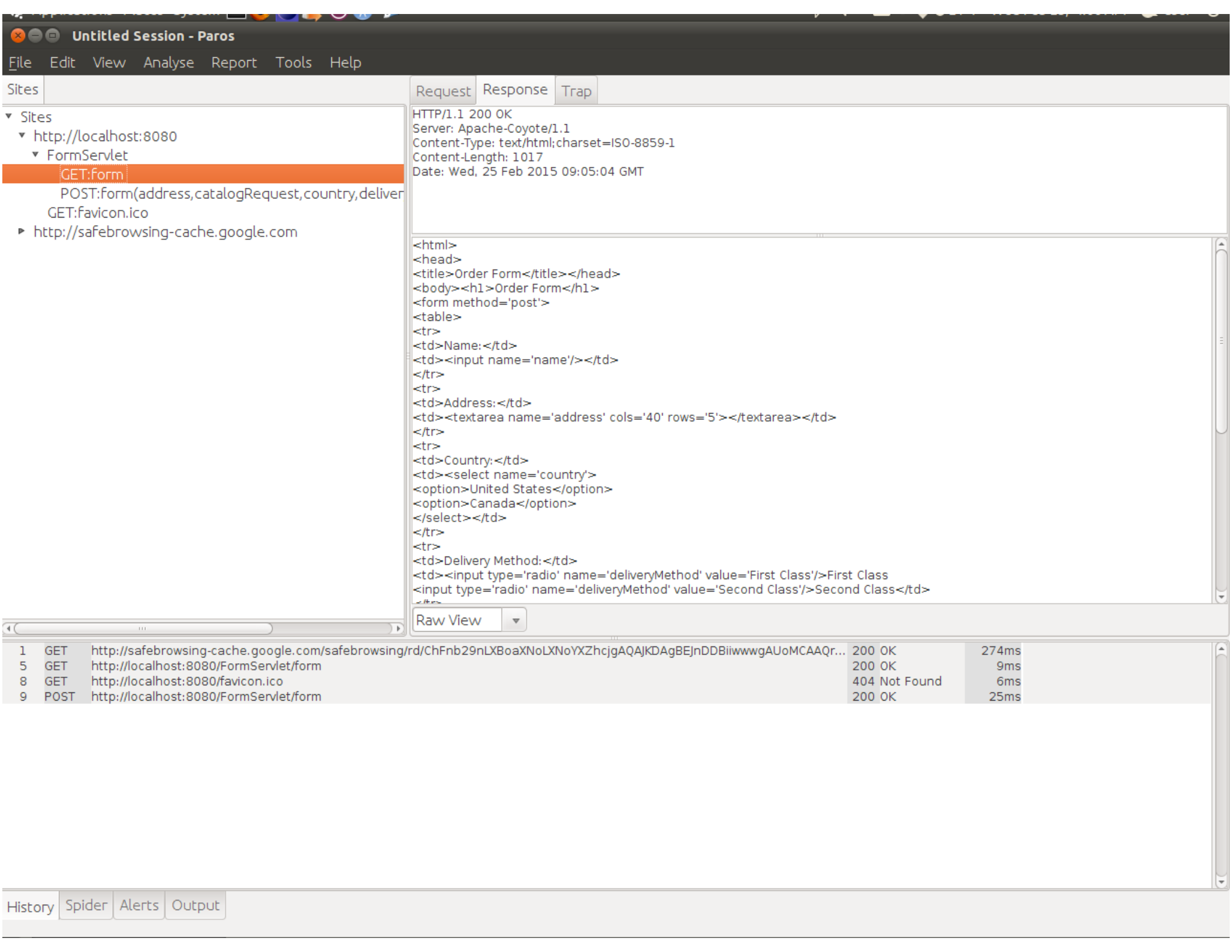


The new **service** method in **HttpServlet** then examines the HTTP method used to send the request (by calling **request.getMethod**) and call one of the following methods: **doGet**, **doPost**, **doHead**, **doPut**, **doTrace**, **doOptions**, and **doDelete**. Each of the seven methods represents an HTTP method. **doGet** and **doPost** are the most often used. As such, you rarely need to override the **service** methods anymore. Instead, you override **doGet** or **doPost** or both **doGet** and **doPost**.

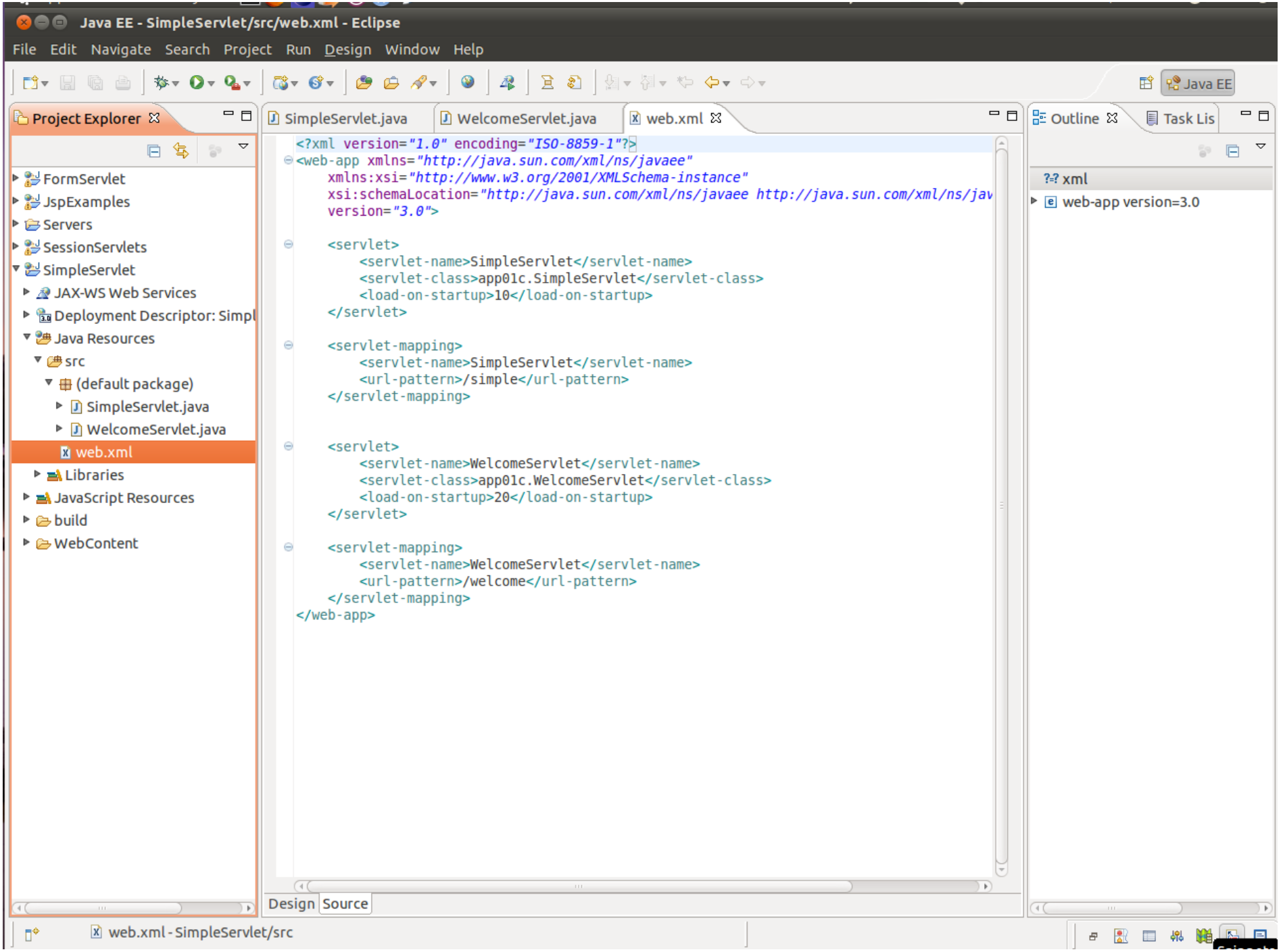


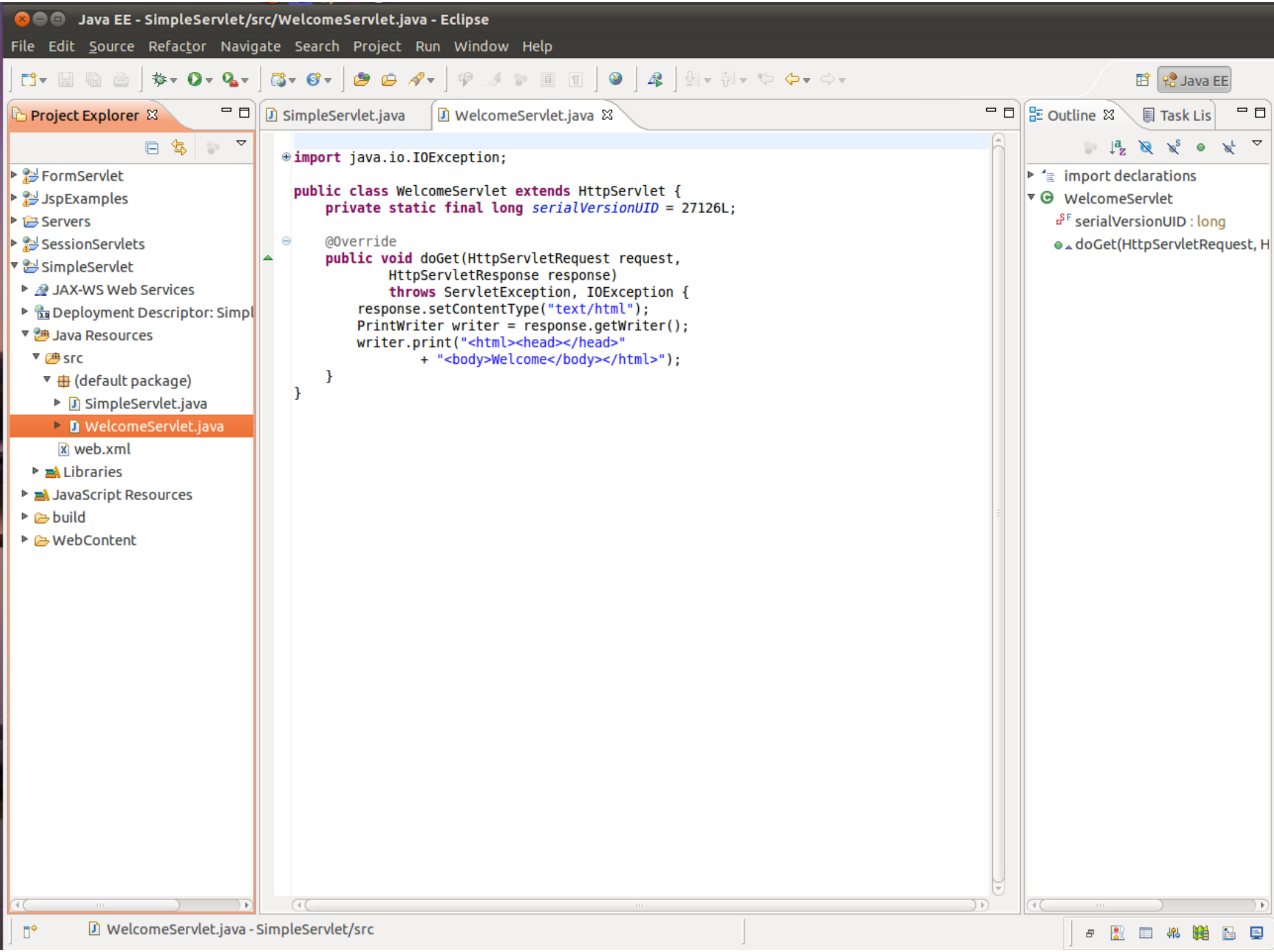

Java EE - FormServlet/src/FormServlet.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
FormServlet
JAX-WS Web Services
Deployment Descriptor: Form
Java Resources
src
(default package)
FormServlet.java
Libraries
JavaScript Resources
build
WebContent
JspExamples
Servers
SessionServlets
SimpleServlet
FormServlet.java
import java.io.IOException;
@WebServlet(name = "FormServlet", urlPatterns = { "/form" })
public class FormServlet extends HttpServlet {
private static final long serialVersionUID = 54L;
private static final String TITLE = "Order Form";
@Override
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter writer = response.getWriter();
writer.println("<html>");
writer.println("<head>");
writer.println("<title>" + TITLE + "</title></head>");
writer.println("<body><h1>" + TITLE + "</h1>");
writer.println("<form method='post'>");
writer.println("<table>");
writer.println("<tr>");
writer.println("<td>Name:</td>");
writer.println("<td><input name='name'/></td>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>Address:</td>");
writer.println("<td><textarea name='address' "
+ "cols='40' rows='5'></textarea></td>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>Country:</td>");
writer.println("<td><select name='country'>");
writer.println("<option>United States</option>");
writer.println("<option>Canada</option>");
writer.println("</select></td>");
writer.println("</tr>");
writer.println("<tr>");
writer.println("<td>Delivery Method:</td>");
writer.println("<td><input type='radio' " +
"name='deliveryMethod'"
+ " value='First Class'/>First Class");
writer.println("<input type='radio' " +
"name='deliveryMethod' "
+ "value='Second Class'/>Second Class</td>");
writer.println("</tr>");
Outline
Task Lis
import declarations
FormServlet
serialVersionUID : long
TITLE : String
doGet(HttpServletRequest, H
doPost(HttpServletRequest,
Run and Trace in Paros
FormServlet.java - FormServlet/src


Order Form - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/FormServlet/form
Google
SWEET Eclipse BlueJ Apache Tomcat W3C W3Schools Liveweave Java Java SE Java EE 6 WebGoat Java Notes MySQL
Order Form
Order Form
Name: My Name
Address: My Address
Country: United States
Shipping Instructions: Please Ship ASAP!
Delivery Method: First Class
Catalog Request: Yes
Debug Info
name: My Name
address: My Address
country: United States
deliveryMethod: First Class
instruction: Please Ship ASAP!
catalogRequest: on
Done


Untitled Session - Paros
File Edit View Analyse Report Tools Help
Sites
Sites
http://localhost:8080
FormServlet
GET:Form
POST:Form(address,catalogRequest,country,deliver
GET:favicon.ico
http://safebrowsing-cache.google.com
Request Response Trap
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 1017
Date: Wed, 25 Feb 2015 09:05:04 GMT
<html>
<head>
<title>Order Form</title></head>
<body><h1>Order Form</h1>
<form method='post'>
<table>
<tr>
<td>Name:</td>
<td><input name='name'/></td>
</tr>
<tr>
<td>Address:</td>
<td><textarea name='address' cols='40' rows='5'></textarea></td>
</tr>
<tr>
<td>Country:</td>
<td><select name='country'>
<option>United States</option>
<option>Canada</option>
</select></td>
</tr>
<tr>
<td>Delivery Method:</td>
<td><input type='radio' name='deliveryMethod' value='First Class'/>First Class
<input type='radio' name='deliveryMethod' value='Second Class'/>Second Class</td>
Raw View
1 GET http://safebrowsing-cache.google.com/safebrowsing/rd/ChFnb29nLXBoaXNoLXNoYXZhcjgAQAJKDAgBEJnDDBiiwwwgAUoMCAAQr... 200 OK 274ms
5 GET http://localhost:8080/FormServlet/form 200 OK 9ms
8 GET http://localhost:8080/favicon.ico 404 Not Found 6ms
9 POST http://localhost:8080/FormServlet/form 200 OK 25ms
History Spider Alerts Output

Servlet Mapping with “web.xml”




Java EE - SimpleServlet/src/WelcomeServlet.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
FormServlet
JspExamples
Servers
SessionServlets
SimpleServlet
JAX-WS Web Services
Deployment Descriptor: Simpl
Java Resources
src
(default package)
SimpleServlet.java
WelcomeServlet.java
web.xml
Libraries
JavaScript Resources
build
WebContent
SimpleServlet.java
WelcomeServlet.java
import java.io.IOException;
public class WelcomeServlet extends HttpServlet {
private static final long serialVersionUID = 27126L;
@Override
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter writer = response.getWriter();
writer.print("<html><head></head>"
+ "<body>Welcome</body></html>");
}
}
Outline
Task Lis
import declarations
WelcomeServlet
serialVersionUID : long
doGet(HttpServletRequest, H
WelcomeServlet.java - SimpleServlet/src

HTTP Session Management

Session Management

Session management or session tracking is a very important topic in web application development. This is due to the fact that HTTP, the language of the web, is stateless. A web server by default does not know if an HTTP request comes from a first time user or from someone who has visited it before.

For example, a webmail application requires its users to log in before they can check their emails. However, once a user types in the correct user name and password, the application should not prompt the user to log in again to access different parts of the application. The application needs to remember which users have successfully logged in. In other words, it must be able to manage user sessions.

This chapter explores four techniques you can use to retain states: URL rewriting, hidden fields, cookies, and the **HttpSession** objects. The samples presented in this chapter are part of the **app02a** application.

URL Rewriting

URL rewriting is a session tracking technique whereby you add a token or multiple tokens as a query string to a URL. The token is generally in key=value format:

```
url?key-1=value-1&key-2=value-2 ... &key-n=value-n
```

Hidden Fields

Using hidden fields to retain states is similar to employing the URL rewriting technique. Instead of appending values to the URL, however, you put them in hidden fields within an HTML form. When the user submits the form, the values in the hidden fields are also passed to the server. Hidden fields are only suitable if your page contains a form or you can add one to it. The advantage of this technique over URL rewriting is you can pass much more characters to the server and no character encoding is necessary. Like URL rewriting, however, this technique is only good if the information to be passed doesn't need to span many pages.

Cookies

URL rewriting and hidden fields are only suitable for retaining information that does not need to span many pages. If the information needs to be carried over more than a few pages, the two techniques become harder to implement because you have to manage the information for every page. Fortunately, cookies can tackle what URL rewriting and hidden fields are not capable of handling.

A cookie is a small piece of information that is passed back and forth between the web server and the browser automatically. Cookies are suitable for information that needs to span many pages. Because cookies are embedded as HTTP headers, the process of transferring them is handled by the HTTP protocol. Apart from that, you can make a cookie live as long or as short as you want. A web browser is expected to support up to twenty cookies per web server.

The downside of cookies is a user can refuse to accept cookies by changing his/her browser settings.

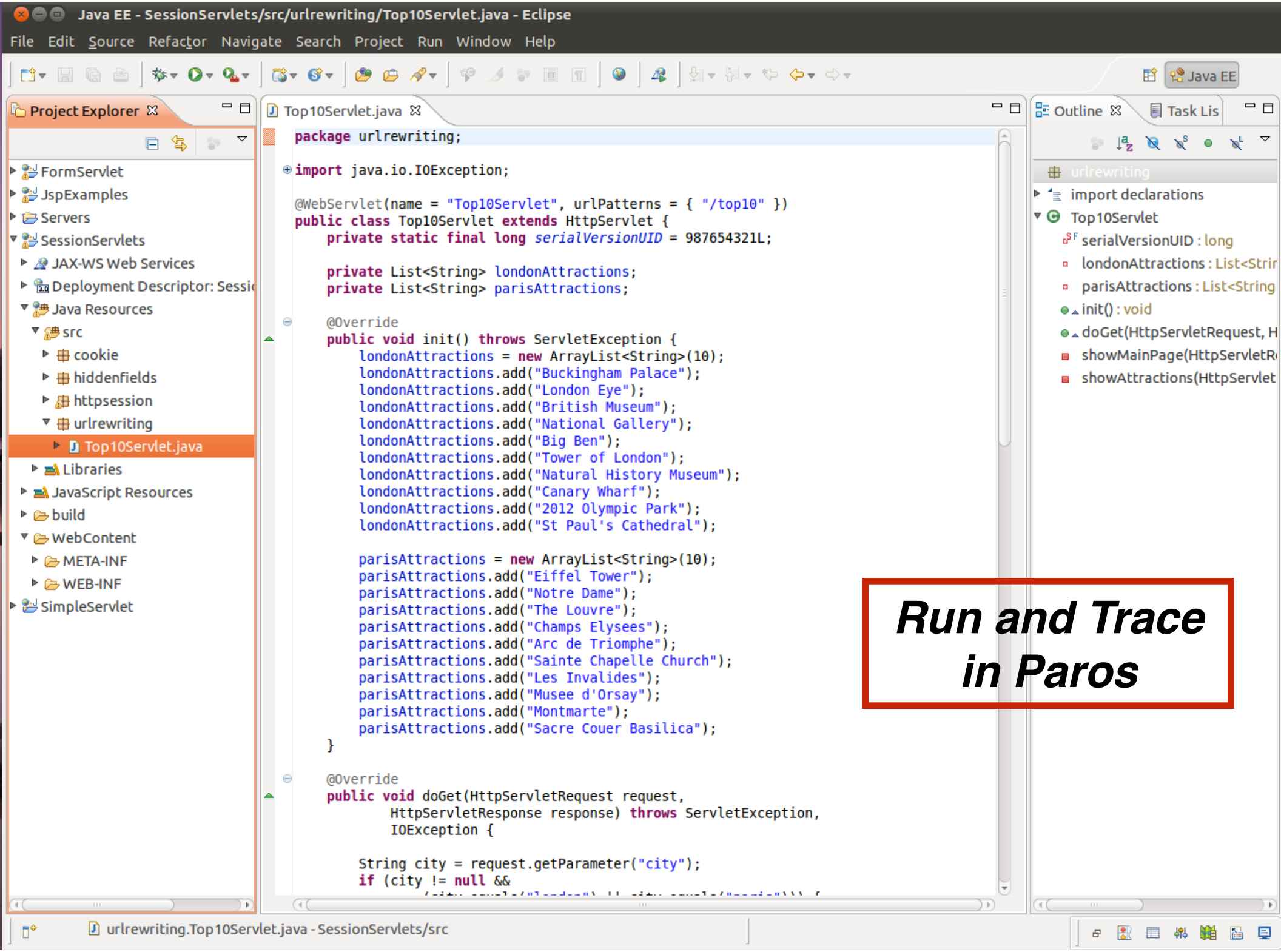
HttpSession Objects

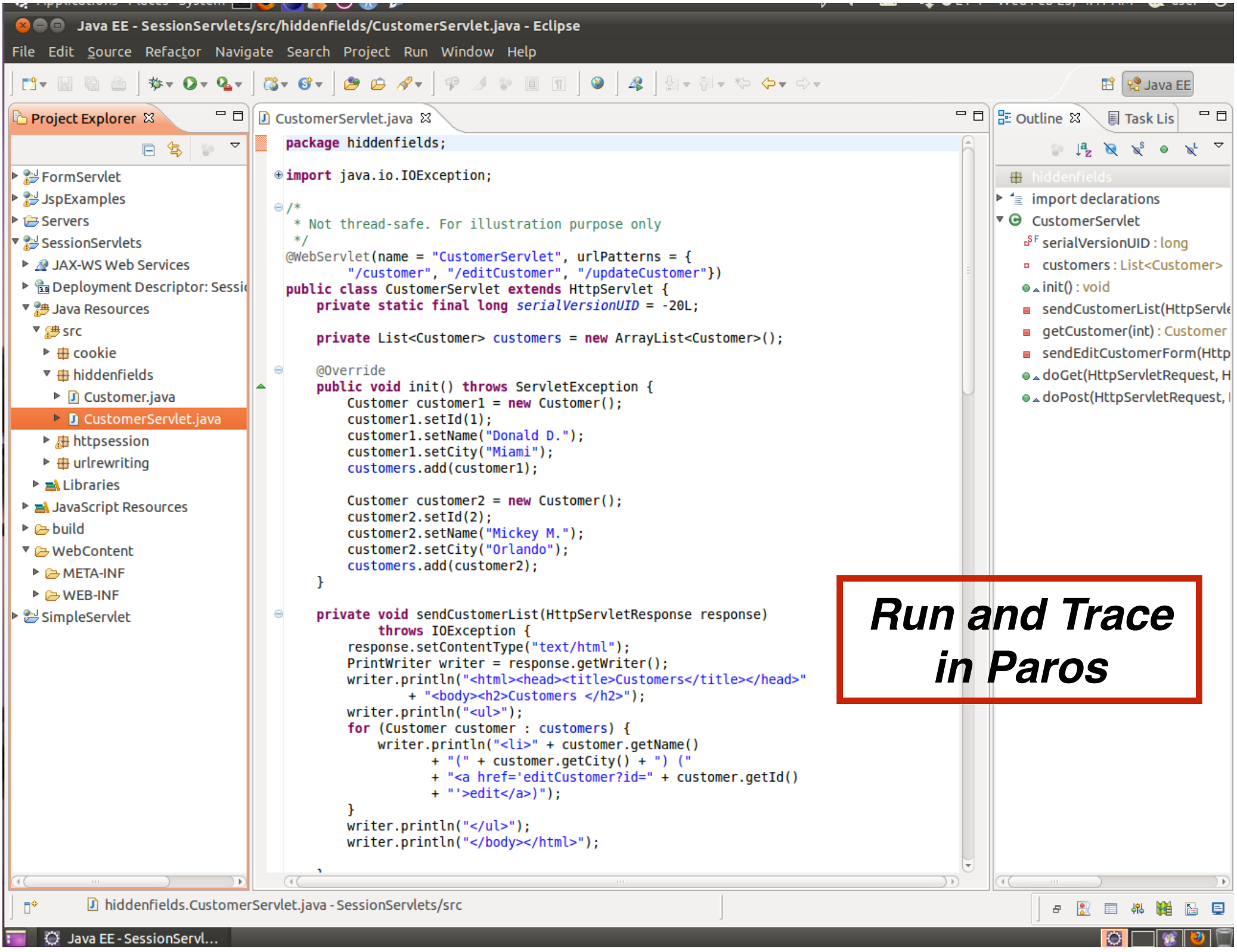
Of all the session tracking techniques, **HttpSession** objects are the most powerful and the most versatile. A user can have zero or one **HttpSession** and can only access his/her own **HttpSession**.

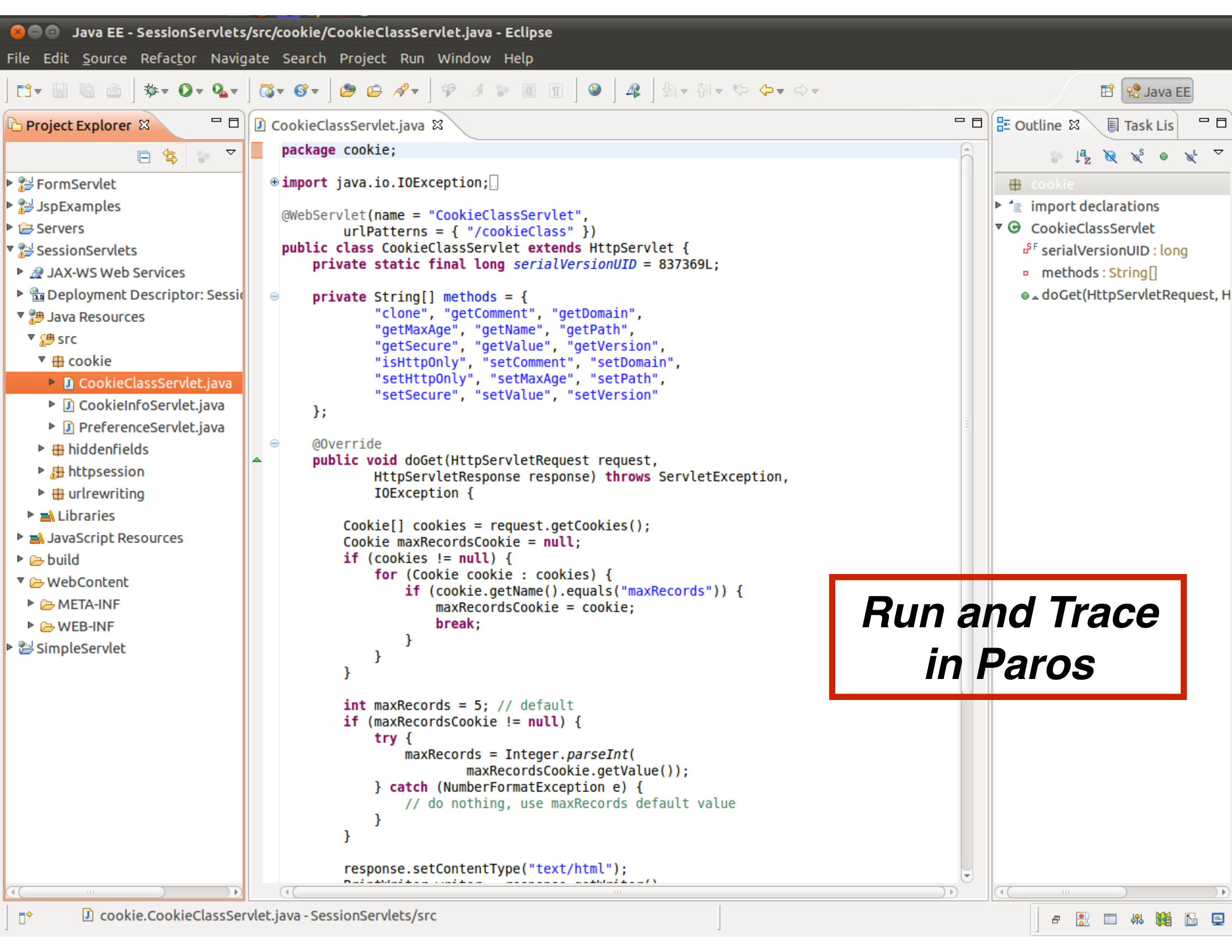
An **HttpSession** is created automatically when a user first visits a site. You retrieve a user's **HttpSession** by calling the **getSession** method on the **HttpServletRequest**. There are two overloads of **getSession**:

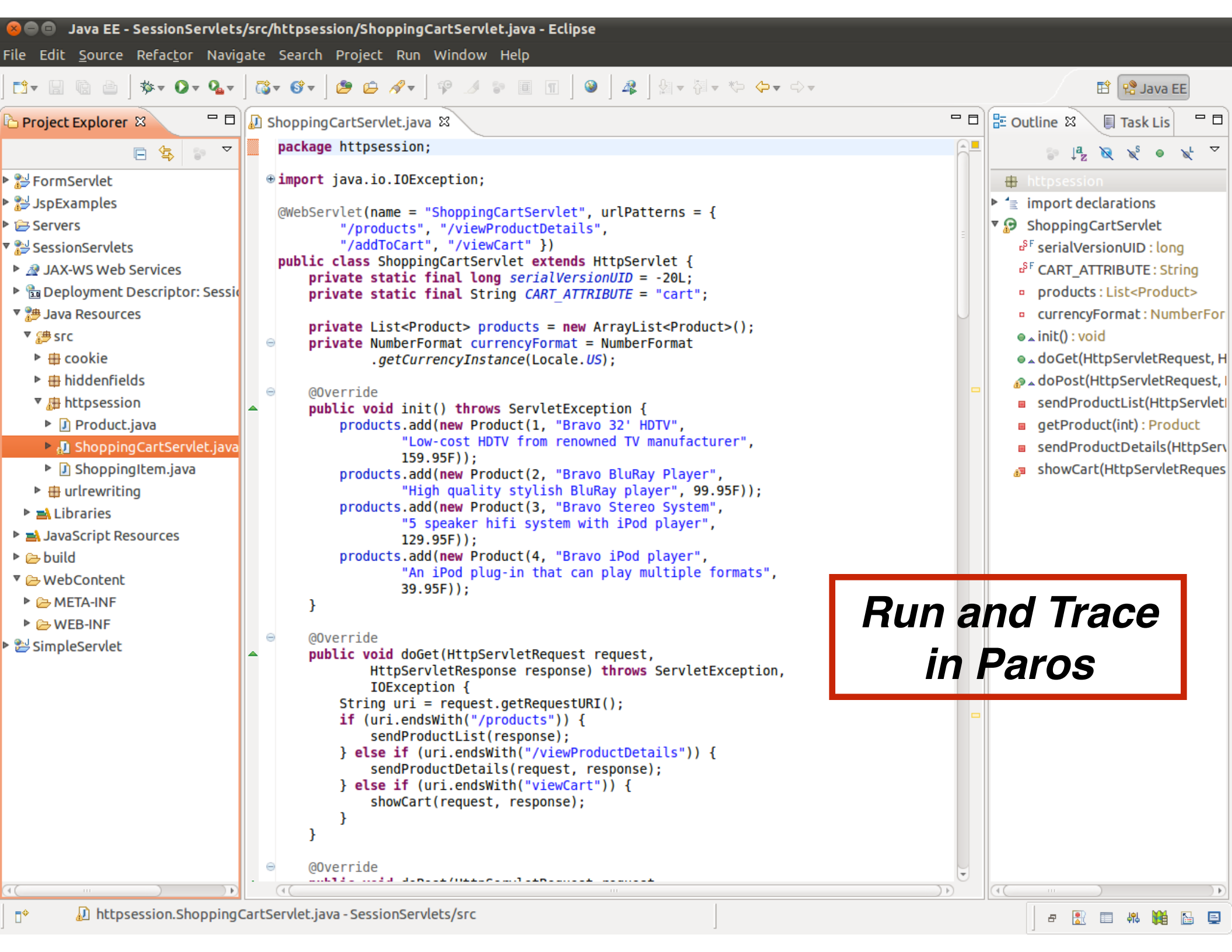
```
HttpSession getSession()
HttpSession getSession(boolean create)
```

Note that unlike in URL rewriting, hidden fields and cookies, a value put in an **HttpSession** is stored in memory. As such, you should only store the smallest possible objects in it and not too many of them. Even though modern servlet containers can move objects in **HttpSessions** to secondary storage when it's about to run out of memory, this would be a performance drag. Therefore, be careful what you store in them.


Java EE - SessionServlets/src/urlrewriting/Top10Servlet.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
FormServlet
JspExamples
Servers
SessionServlets
JAX-WS Web Services
Deployment Descriptor: Sessi
Java Resources
src
cookie
hiddenfields
httpsession
urlrewriting
Top10Servlet.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
SimpleServlet
Top10Servlet.java
package urlrewriting;
import java.io.IOException;
@WebServlet(name = "Top10Servlet", urlPatterns = { "/top10" })
public class Top10Servlet extends HttpServlet {
private static final long serialVersionUID = 987654321L;
private List<String> londonAttractions;
private List<String> parisAttractions;
@Override
public void init() throws ServletException {
londonAttractions = new ArrayList<String>(10);
londonAttractions.add("Buckingham Palace");
londonAttractions.add("London Eye");
londonAttractions.add("British Museum");
londonAttractions.add("National Gallery");
londonAttractions.add("Big Ben");
londonAttractions.add("Tower of London");
londonAttractions.add("Natural History Museum");
londonAttractions.add("Canary Wharf");
londonAttractions.add("2012 Olympic Park");
londonAttractions.add("St Paul's Cathedral");
parisAttractions = new ArrayList<String>(10);
parisAttractions.add("Eiffel Tower");
parisAttractions.add("Notre Dame");
parisAttractions.add("The Louvre");
parisAttractions.add("Champs Elysees");
parisAttractions.add("Arc de Triomphe");
parisAttractions.add("Sainte Chapelle Church");
parisAttractions.add("Les Invalides");
parisAttractions.add("Musee d'Orsay");
parisAttractions.add("Montmarte");
parisAttractions.add("Sacre Couer Basilica");
}
@Override
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
String city = request.getParameter("city");
if (city != null &&
Outline
Task Lis
urlrewriting
import declarations
Top10Servlet
serialVersionUID : long
londonAttractions : List<Strir
parisAttractions : List<String
init() : void
doGet(HttpServletRequest, H
showMainPage(HttpServletR
showAttractions(HttpServlet
Run and Trace in Paros
urlrewriting.Top10Servlet.java - SessionServlets/src


Java EE - SessionServlets/src/hiddenfields/CustomerServlet.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
FormServlet
JspExamples
Servers
SessionServlets
JAX-WS Web Services
Deployment Descriptor: Sessi
Java Resources
src
cookie
hiddenfields
Customer.java
CustomerServlet.java
httpsession
urlrewriting
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
SimpleServlet
CustomerServlet.java
package hiddenfields;
import java.io.IOException;
/*
* Not thread-safe. For illustration purpose only
*/
@WebServlet(name = "CustomerServlet", urlPatterns = {
"/customer", "/editCustomer", "/updateCustomer"})
public class CustomerServlet extends HttpServlet {
private static final long serialVersionUID = -20L;
private List<Customer> customers = new ArrayList<Customer>();
@Override
public void init() throws ServletException {
Customer customer1 = new Customer();
customer1.setId(1);
customer1.setName("Donald D.");
customer1.setCity("Miami");
customers.add(customer1);
Customer customer2 = new Customer();
customer2.setId(2);
customer2.setName("Mickey M.");
customer2.setCity("Orlando");
customers.add(customer2);
}
private void sendCustomerList(HttpServletResponse response)
throws IOException {
response.setContentType("text/html");
PrintWriter writer = response.getWriter();
writer.println("<html><head><title>Customers</title></head>"
+ "<body><h2>Customers </h2>");
writer.println("<ul>");
for (Customer customer : customers) {
writer.println("<li>" + customer.getName()
+ "(" + customer.getCity() + ") ("
+ "<a href='editCustomer?id=" + customer.getId()
+ "'>edit)");
}
writer.println("
");
writer.println("</body></html>");
Outline
Task Lis
hiddenfields
import declarations
CustomerServlet
serialVersionUID : long
customers : List<Customer>
init() : void
sendCustomerList(HttpServle
getCustomer(int) : Customer
sendEditCustomerForm(Http
doGet(HttpServletRequest, H
doPost(HttpServletRequest,
Run and Trace in Paros
hiddenfields.CustomerServlet.java - SessionServlets/src
Java EE - SessionServl...


Java EE - SessionServlets/src/cookie/CookieClassServlet.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
FormServlet
JspExamples
Servers
SessionServlets
JAX-WS Web Services
Deployment Descriptor: Sessi
Java Resources
src
cookie
CookieClassServlet.java
CookieInfoServlet.java
PreferenceServlet.java
hiddenfields
httpsession
urlrewriting
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
SimpleServlet
CookieClassServlet.java
package cookie;
import java.io.IOException;
@WebServlet(name = "CookieClassServlet",
urlPatterns = { "/cookieClass" })
public class CookieClassServlet extends HttpServlet {
private static final long serialVersionUID = 837369L;
private String[] methods = {
"clone", "getComment", "getDomain",
"getMaxAge", "getName", "getPath",
"getSecure", "getValue", "getVersion",
"isHttpOnly", "setComment", "setDomain",
"setHttpOnly", "setMaxAge", "setPath",
"setSecure", "setValue", "setVersion"
};
@Override
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
Cookie[] cookies = request.getCookies();
Cookie maxRecordsCookie = null;
if (cookies != null) {
for (Cookie cookie : cookies) {
if (cookie.getName().equals("maxRecords")) {
maxRecordsCookie = cookie;
break;
}
}
}
int maxRecords = 5; // default
if (maxRecordsCookie != null) {
try {
maxRecords = Integer.parseInt(
maxRecordsCookie.getValue());
} catch (NumberFormatException e) {
// do nothing, use maxRecords default value
}
}
response.setContentType("text/html");
Outline
Task Lis
cookie
import declarations
CookieClassServlet
serialVersionUID : long
methods : String[]
doGet(HttpServletRequest, H
Run and Trace in Paros
cookie.CookieClassServlet.java - SessionServlets/src


Java EE - SessionServlets/src/httpsession/ShoppingCartServlet.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
FormServlet
JspExamples
Servers
SessionServlets
JAX-WS Web Services
Deployment Descriptor: Sessi
Java Resources
src
cookie
hiddenfields
httpsession
Product.java
ShoppingCartServlet.java
ShoppingItem.java
urlrewriting
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
SimpleServlet
ShoppingCartServlet.java
package httpsession;
import java.io.IOException;
@WebServlet(name = "ShoppingCartServlet", urlPatterns = {
"/products", "/viewProductDetails",
"/addToCart", "/viewCart" })
public class ShoppingCartServlet extends HttpServlet {
private static final long serialVersionUID = -20L;
private static final String CART_ATTRIBUTE = "cart";
private List<Product> products = new ArrayList<Product>();
private NumberFormat currencyFormat = NumberFormat
.getCurrencyInstance(Locale.US);
@Override
public void init() throws ServletException {
products.add(new Product(1, "Bravo 32' HDTV",
"Low-cost HDTV from renowned TV manufacturer",
159.95F));
products.add(new Product(2, "Bravo BluRay Player",
"High quality stylish BluRay player", 99.95F));
products.add(new Product(3, "Bravo Stereo System",
"5 speaker hifi system with iPod player",
129.95F));
products.add(new Product(4, "Bravo iPod player",
"An iPod plug-in that can play multiple formats",
39.95F));
}
@Override
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
String uri = request.getRequestURI();
if (uri.endsWith("/products")) {
sendProductList(response);
} else if (uri.endsWith("/viewProductDetails")) {
sendProductDetails(request, response);
} else if (uri.endsWith("viewCart")) {
showCart(request, response);
}
}
@Override
Outline
Task Lis
httpsession
import declarations
ShoppingCartServlet
serialVersionUID : long
CART_ATTRIBUTE : String
products : List<Product>
currencyFormat : NumberFor
init() : void
doGet(HttpServletRequest, H
doPost(HttpServletRequest,
sendProductList(HttpServlet
getProduct(int) : Product
sendProductDetails(HttpServ
showCart(HttpServletReques
Run and Trace in Paros
httpsession.ShoppingCartServlet.java - SessionServlets/src

Java Server Pages (JSP)

An Overview of JSP

A JSP page is essentially a servlet. However, working with JSP pages is easier than with servlets for two reasons. First, you do not have to compile JSP pages. Second, JSP pages are basically text files with **jsp** extension and you can use any text editor to write them.

JSP pages run on a JSP container. A servlet container is normally also a JSP container. Tomcat, for instance, is a servlet/JSP container.

The first time a JSP page is requested, a servlet/JSP container does two things:

1. Translate the JSP page into a JSP page implementation class, which is a Java class that implements the **javax.servlet.jsp.JspPage** interface or its subinterface **javax.servlet.jsp.HttpJspPage**. **JspPage** is a subinterface of **javax.servlet.Servlet** and this makes every JSP page a servlet. The class name of the generated servlet is dependent on the servlet/JSP container. You do not have to worry about this because you do not have to work with it directly. If there is a translation error, an error message will be sent to the client.

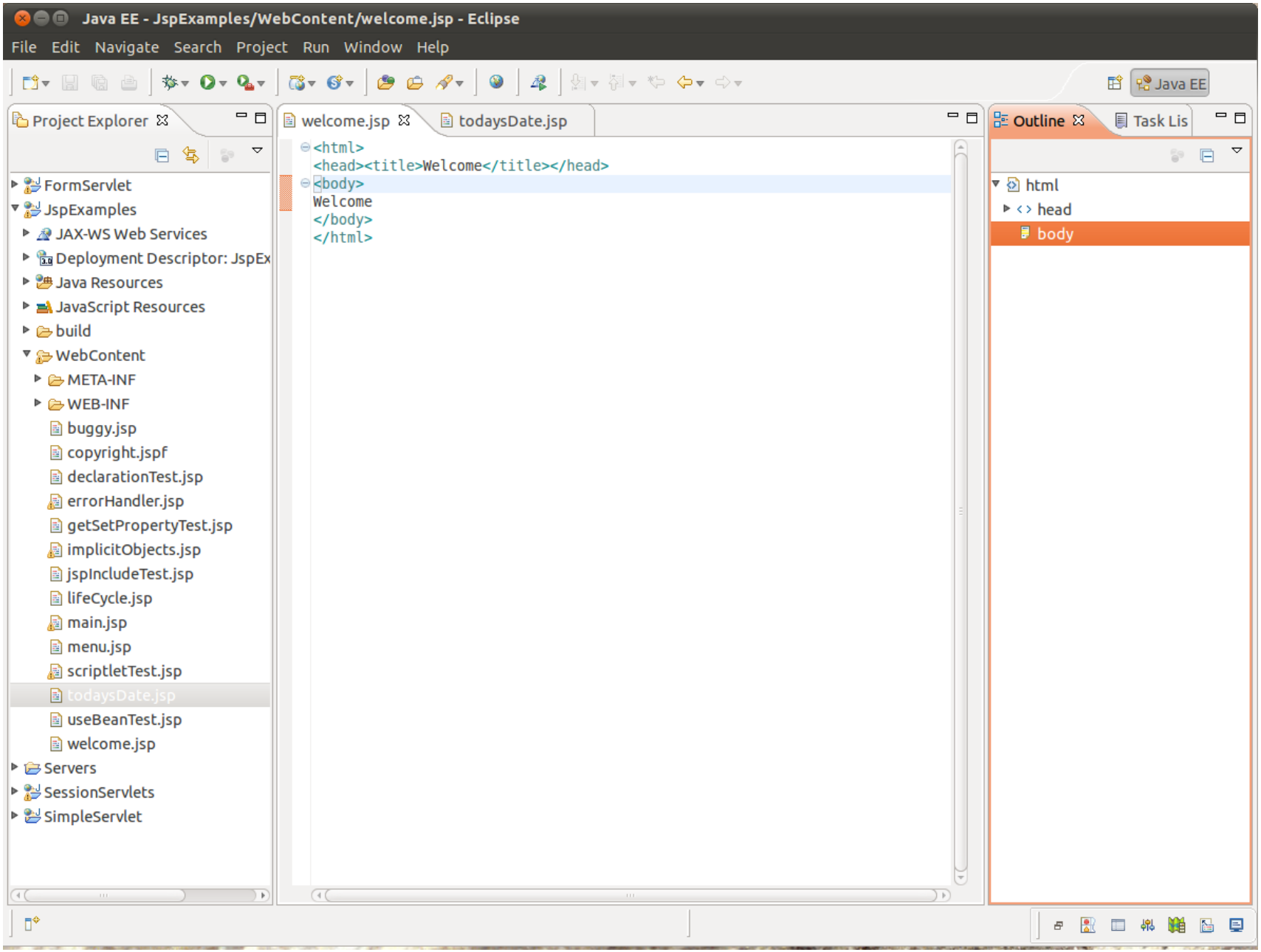
2. If the translation was successful, the servlet/JSP container compiles the servlet class. The container then loads and instantiates the Java bytecode as well as performs the lifecycle operations it normally does a servlet.

Java Server Pages (JSP)

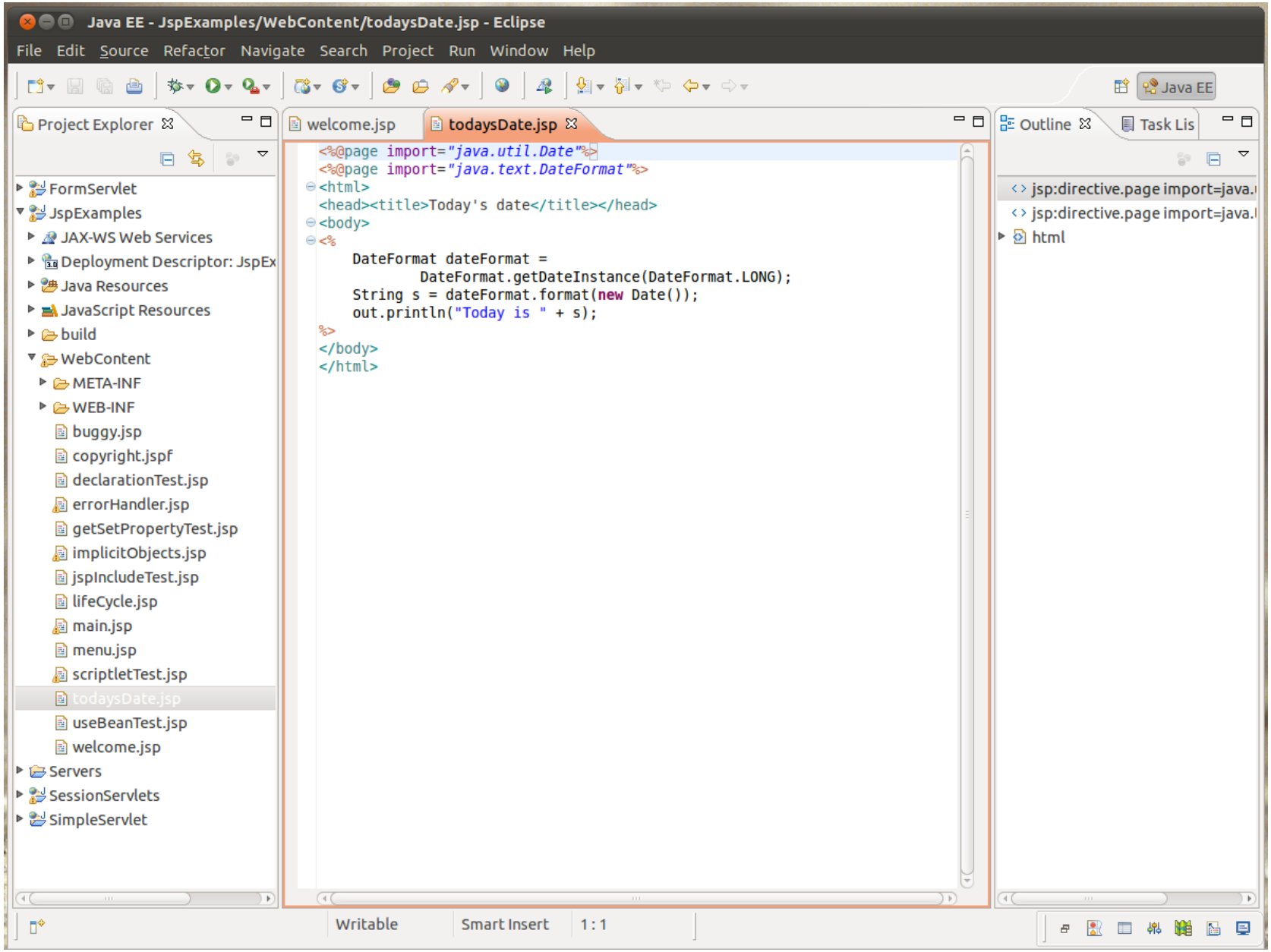
For subsequent requests for the same JSP page, the servlet/JSP container checks if the JSP page has been modified since the last time it was translated. If so, it will be retranslated, re-compiled, and executed. If not, the JSP servlet already in memory is executed. This way, the first invocation of a JSP page always takes longer than subsequent requests because it involves translation and compilation. To get around this problem, you can do one of the following:

A JSP page can contain template data and syntactic elements. An element is something with a special meaning to the JSP translator. For example, **<%** is an element because it denotes the start of a Java code block within a JSP page. **%>** is also an element because it terminates a Java code block. Everything else that is not an element is template data. Template data is sent as is to the browser. For instance, HTML tags and text in a JSP page are template data.

JSP Example - All HTML



JSP Example - HTML + Java

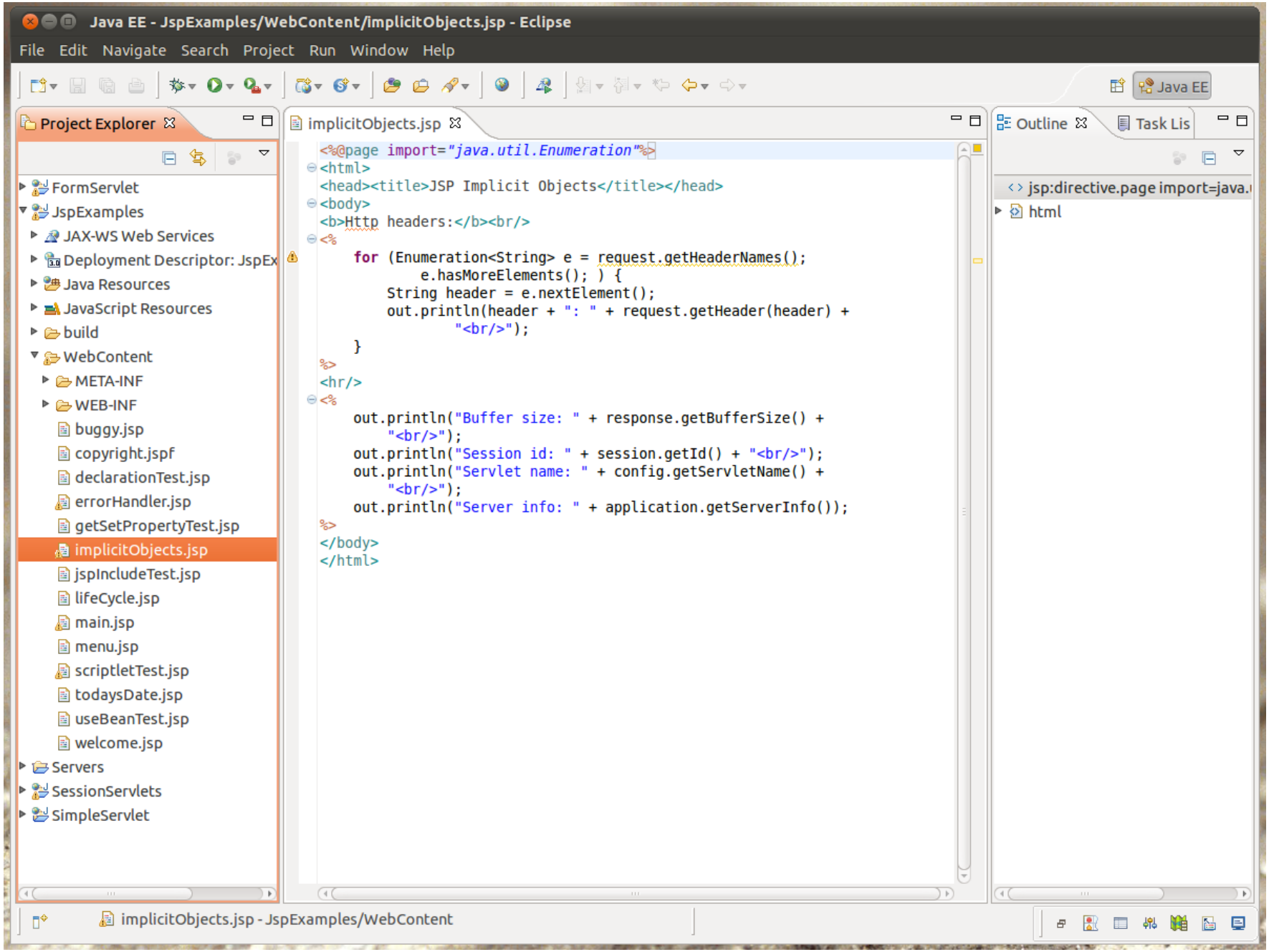


Implied Objects in JSP

In JSP you can retrieve those objects by using implicit objects. Table 3.1 lists the implicit objects.

Object	Type
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
out	javax.servlet.jsp.JspWriter
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
config	javax.servlet.ServletConfig
pageContext	javax.servlet.jsp.PageContext
page	javax.servlet.jsp.HttpJspPage
exception	java.lang.Throwable

JSP Example - Implicit Objects



JSP - Page Directive

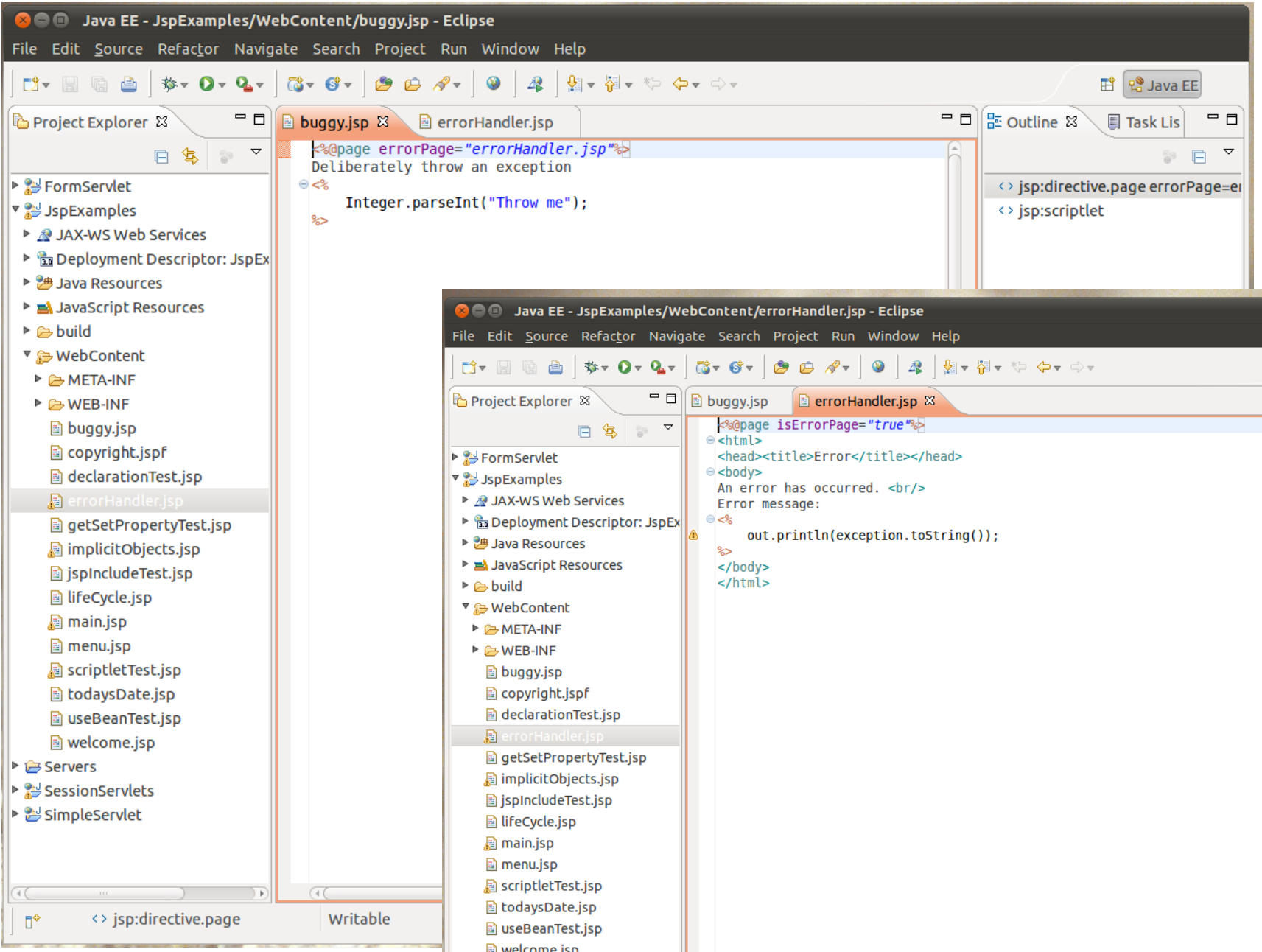
The page Directive

You use the **page** directive to instruct the JSP translator on certain aspects of the current JSP page. For example, you can tell the JSP translator the size of the buffer that should be used for the **out** implicit object, what content type to use, what Java types to import, and so on.

The **page** directive has the following syntax:

```
<%@ page attribute1="value1" attribute2="value2" ... %>
```

The space between @ and **page** is optional and *attribute1*, *attribute2*, and so on are the **page** directive's attributes. Here is the list of attributes for the **page** directive.


Java EE - JspExamples/WebContent/buggy.jsp - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
buggy.jsp
errorHandler.jsp
Outline
Task Lis
<%@page errorPage="errorHandler.jsp"%>
Deliberately throw an exception
<%
Integer.parseInt("Throw me");
%>
jsp:directive.page errorPage=e
jsp:scriptlet
FormServlet
JspExamples
JAX-WS Web Services
Deployment Descriptor: JspEx
Java Resources
JavaScript Resources
build
WebContent
META-INF
WEB-INF
buggy.jsp
copyright.jspf
declarationTest.jsp
errorHandler.jsp
getSetPropertyTest.jsp
implicitObjects.jsp
jspIncludeTest.jsp
lifeCycle.jsp
main.jsp
menu.jsp
scriptletTest.jsp
todaysDate.jsp
useBeanTest.jsp
welcome.jsp
Servers
SessionServlets
SimpleServlet
jsp:directive.page
Writable

Java EE - JspExamples/WebContent/errorHandler.jsp - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
buggy.jsp
errorHandler.jsp
Outline
Task Lis
<%@page isErrorPage="true"%>
<html>
<head><title>Error</title></head>
<body>
An error has occurred.
Error message:
<%
out.println(exception.toString());
%>
</body>
</html>
jsp:directive.page isErrorPage=
html
FormServlet
JspExamples
JAX-WS Web Services
Deployment Descriptor: JspEx
Java Resources
JavaScript Resources
build
WebContent
META-INF
WEB-INF
buggy.jsp
copyright.jspf
declarationTest.jsp
errorHandler.jsp
getSetPropertyTest.jsp
implicitObjects.jsp
jspIncludeTest.jsp
lifeCycle.jsp
main.jsp
menu.jsp
scriptletTest.jsp
todaysDate.jsp
useBeanTest.jsp
welcome.jsp
Servers
SessionServlets
SimpleServlet
Writable
Smart Insert
1 : 1

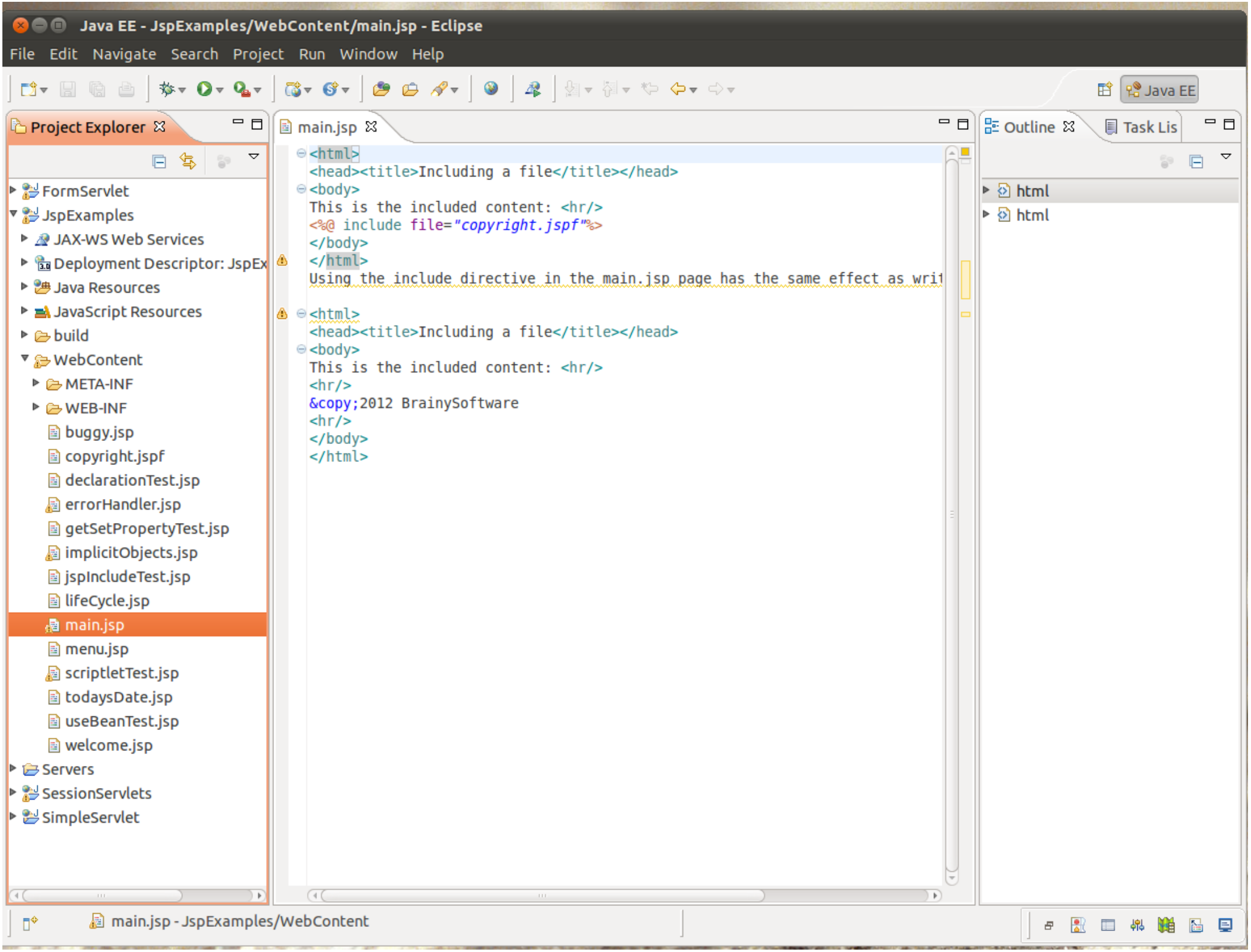
JSP Include Directive

The include Directive

You use the **include** directive to include the content of another file in the current JSP page. You can use multiple **include** directives in a JSP page. Modularizing a particular content into an include file is useful if that content is used by different pages or used by a page in different places.

The syntax of the **include** directive is as follows:

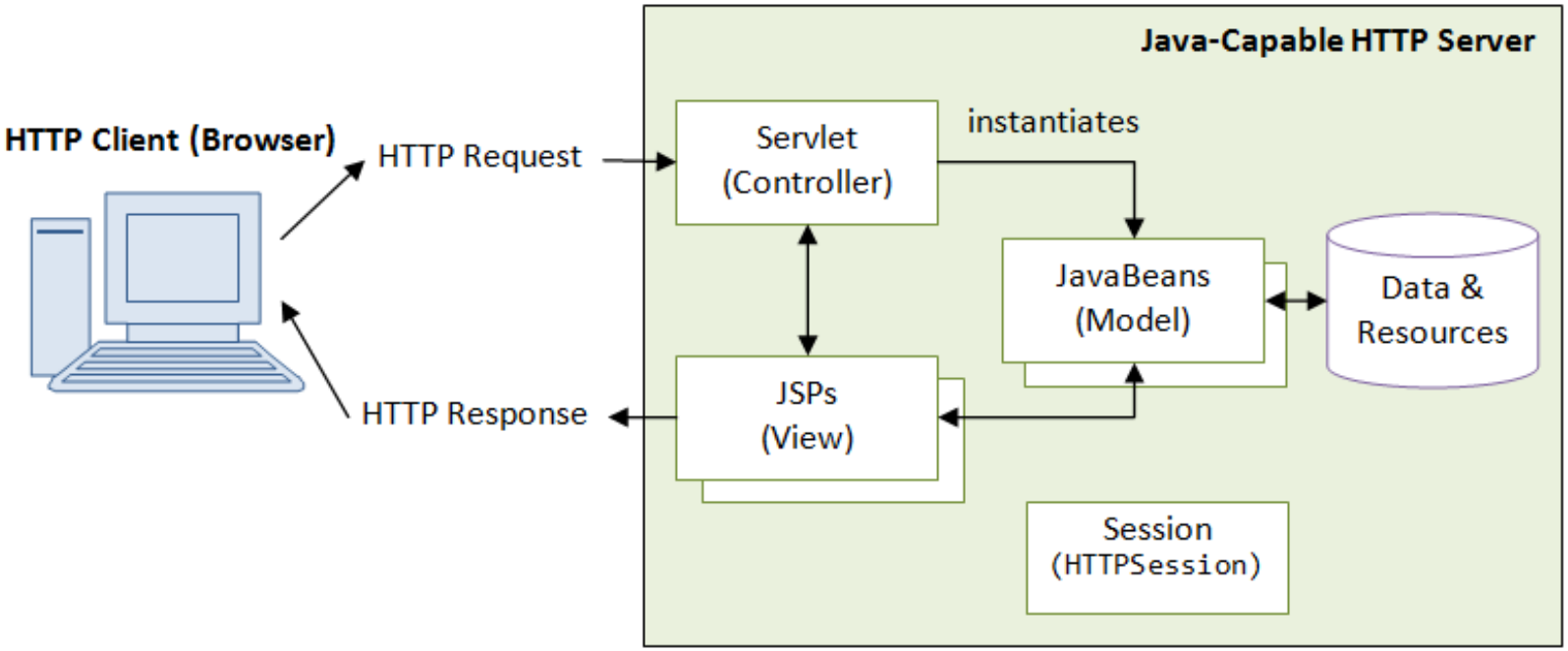
```
<%@ include file="url"%>
```


Java EE - JspExamples/WebContent/main.jsp - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
FormServlet
JspExamples
JAX-WS Web Services
Deployment Descriptor: JspEx
Java Resources
JavaScript Resources
build
WebContent
META-INF
WEB-INF
buggy.jsp
copyright.jspf
declarationTest.jsp
errorHandler.jsp
getSetPropertyTest.jsp
implicitObjects.jsp
jspIncludeTest.jsp
lifeCycle.jsp
main.jsp
menu.jsp
scriptletTest.jsp
todaysDate.jsp
useBeanTest.jsp
welcome.jsp
Servers
SessionServlets
SimpleServlet
main.jsp
<html>
<head><title>Including a file</title></head>
<body>
This is the included content: <hr/>
<%@ include file="copyright.jspf"%>
</body>
</html>
Using the include directive in the main.jsp page has the same effect as wri
<html>
<head><title>Including a file</title></head>
<body>
This is the included content: <hr/>
<hr/>
©2012 BrainySoftware
<hr/>
</body>
</html>
Outline
Task Lis
html
html
main.jsp - JspExamples/WebContent

JSP Book Store

https://www3.ntu.edu.sg/home/ehchua/programming/index.html

JSP / MySQL Bookstore



https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaWebDBApp.html

Java Web Database Applications

Database Connection Pooling in Tomcat with MySQL

Reference: "Apache Tomcat 7.0 JNDI Datasource How-To" @ http://tomcat.apache.org/tomcat-7.0-doc/jndi-datasource-examples-howto.html.

I shall assume that you are familiar with Tomcat and MySQL. Otherwise, read "How to Install and Get Started with Tomcat" and "How to Install and Get Started with MySQL". I shall denote Tomcat's and MySQL's installed directory as $CATALINA_HOME and $MYSQL_HOME respestively.

Creating a new *database connection* for each client request is grossly inefficient, due to the high overhead involved in opening/maintaining/closing the connection. Instead, we usually set up a *database connection pool*, which maintains a number of database connections. A server-side program, in response to a client request, picks up an already-created connection from the pool. After the request is completed, the connection is not closed, but returned to the pool to handle further request.

To configure Tomcat/MySQL for supporting Database Connection Pooling (DBCP):

Step 1: MySQL Database Driver - Copy the MySQL Connector/J JDBC Driver's jar-file (e.g., `mysql-connector-java-5.1.xx-bin.jar`) into Tomcat's "`lib`", i.e., `$CATALINA_HOME/lib`.

Step 2: Create Web Application - Let's start a new web application called "`hellodbcp`" to test database connection pooling. Create a directory "`hellodbcp`" under `$CATALINA_HOME/webapps`. Create sub-directories "`WEB-INF`" and "`META-INF`" under the context root "`hellodbcp`". Create sub-sub-directories "`src`", "`classes`", and "`lib`" under "`WEB-INF`".

Step 3: Configure JNDI Datasource - Next, we shall configure the JNDI (Java Naming and Directory Interface) Datasource in "`context.xml`". For application-specific configuration, save it under "`webapps\hellodbcp\META-INF`". For server-wide configuration, put the `<Resource>` element under `<GlobalNamingResources>` in `$CATALINA_HOME\conf\server.xml`.

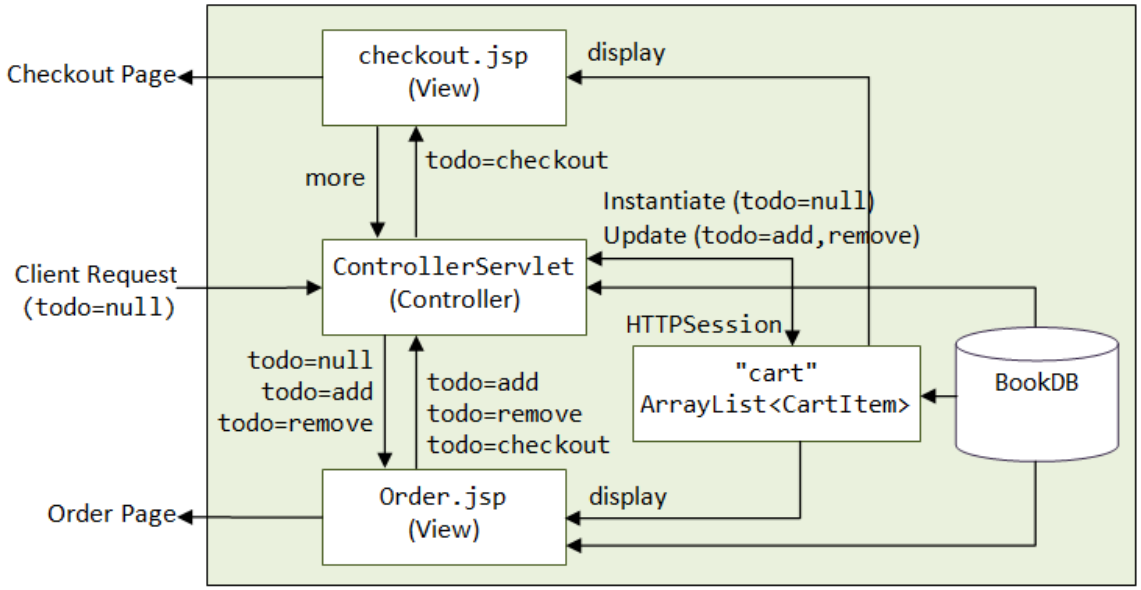
```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<Context reloadable="true">
  <!--
    maxActive: Maximum number of dB connections in pool. Set to -1 for no limit.
    maxIdle: Maximum number of idle dB connections to retain in pool. Set to -1 for no limit.
    maxWait: Maximum milliseconds to wait for a dB connection to become available
             Set to -1 to wait indefinitely.
  -->
  <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
     maxActive="100" maxIdle="30" maxWait="10000" removeAbandoned="true"
     username="myuser" password="xxxx" driverClassName="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost:3306/ebookshop" />
</Context>
```

The above configuration declares a *JNDI resource name* called "`jdbc/TestDB`" corresponds to the MySQL connection "`mysql://localhost:3306/ebookshop`".

JSP Bookstore

https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaWebDBApp.html

- ***In-class Lab Tasks:***
  1. ***Create Eclipse project***
  2. ***Setup MySQL***
  3. ***Create & Run JSP Bookstore***



Java SE Security

http://docs.oracle.com/javase/tutorial/security/index.html

Trail: Security Features in Java SE

In this trail you'll learn how the built-in Java™ security features protect you from malevolent programs. You'll see how to use tools to control access to resources, to generate and to check digital signatures, and to create and to manage keys needed for signature generation and checking. You'll also see how to incorporate cryptography services, such as digital signature generation and checking, into your programs.

The security features provided by the Java Development Kit (JDK™) are intended for a variety of audiences:

- **Users running programs**:

  Built-in security functionality protects you from malevolent programs (including viruses), maintains the privacy of your files and information about you, and authenticates the identity of each code provider. You can subject applications and applets to security controls when you need to.

- **Developers**:

  You can use API methods to incorporate security functionality into your programs, including cryptography services and security checks. The API framework enables you to define and integrate your own permissions (controlling access to specific resources), cryptography service implementations, security manager implementations, and policy implementations. In addition, classes are provided for management of your public/private key pairs and public key certificates from people you trust.

- **Systems administrators, developers, and users**:

  JDK tools manage your keystore (database of keys and certificates); generate digital signatures for JAR files, and verify the authenticity of such signatures and the integrity of the signed contents; and create and modify the policy files that define your installation's security policy.

Java is "Type Safe"

Java™ Security Overview

1 Introduction

The Java™ platform was designed with a strong emphasis on security. At its core, the Java language itself is type-safe and provides automatic garbage collection, enhancing the robustness of application code. A secure class loading and verification mechanism ensures that only legitimate Java code is executed.

The initial version of the Java platform created a safe environment for running potentially untrusted code, such as Java applets downloaded from a public network. As the platform has grown and widened its range of deployment, the Java security architecture has correspondingly evolved to support an increasing set of services. Today the architecture includes a large set of application programming interfaces (APIs), tools, and implementations of commonly-used security algorithms, mechanisms, and protocols. This provides the developer a comprehensive security framework for writing applications, and also provides the user or administrator a set of tools to securely manage applications.

The Java security APIs span a wide range of areas. Cryptographic and public key infrastructure (PKI) interfaces provide the underlying basis for developing secure applications. Interfaces for performing authentication and access control enable applications to guard against unauthorized access to protected resources.

The APIs allow for multiple interoperable implementations of algorithms and other security services. Services are implemented in *providers*, which are plugged into the Java platform via a standard interface that makes it easy for applications to obtain security services without having to know anything about their implementations. This allows developers to focus on how to integrate security into their applications, rather than on how to actually implement complex security mechanisms.

The Java platform includes a number of providers that implement a core set of security services. It also allows for additional custom providers to be installed. This enables developers to extend the platform with new security mechanisms.

This paper gives a broad overview of security in the Java platform, from secure language features to the security APIs, tools, and built-in provider services, highlighting key packages and classes where applicable. Note that this paper is based on Java™ SE version 8.

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

Byte Code Verification & Class Level Security

2 Java Language Security and Bytecode Verification

The Java language is designed to be type-safe and easy to use. It provides automatic memory management, garbage collection, and range-checking on arrays. This reduces the overall programming burden placed on developers, leading to fewer subtle programming errors and to safer, more robust code.

In addition, the Java language defines different access modifiers that can be assigned to Java classes, methods, and fields, enabling developers to restrict access to their class implementations as appropriate. Specifically, the language defines four distinct access levels: `private`, `protected`, `public`, and, if unspecified, `package`. The most open access specifier is `public` access is allowed to anyone. The most restrictive modifier is `private` access is not allowed outside the particular class in which the private member (a method, for example) is defined. The `protected` modifier allows access to any subclass, or to other classes within the same package. Package-level access only allows access to classes within the same package.

A compiler translates Java programs into a machine-independent bytecode representation. A bytecode verifier is invoked to ensure that only legitimate bytecodes are executed in the Java runtime. It checks that the bytecodes conform to the Java Language Specification and do not violate Java language rules or namespace restrictions. The verifier also checks for memory management violations, stack underflows or overflows, and illegal data typecasts. Once bytecodes have been verified, the Java runtime prepares them for execution.

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

Java Security APIs

3 Basic Security Architecture

The Java platform defines a set of APIs spanning major security areas, including cryptography, public key infrastructure, authentication, secure communication, and access control. These APIs allow developers to easily integrate security into their application code. They were designed around the following principles:

Implementation independence

Applications do not need to implement security themselves. Rather, they can request security services from the Java platform. Security services are implemented in providers (see below), which are plugged into the Java platform via a standard interface. An application may rely on multiple independent providers for security functionality.

Implementation interoperability

Providers are interoperable across applications. Specifically, an application is not bound to a specific provider, and a provider is not bound to a specific application.

Algorithm extensibility

The Java platform includes a number of built-in providers that implement a basic set of security services that are widely used today. However, some applications may rely on emerging standards not yet implemented, or on proprietary services. The Java platform supports the installation of custom providers that implement such services.

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

Security Providers

Applications rely on the relevant `getInstance` method to obtain a security service from an underlying provider. For example, message digest creation represents one type of service available from providers. (Section 4 discusses message digests and other cryptographic services.) An application invokes the `getInstance` method in the `java.security.MessageDigest` class to obtain an implementation of a specific message digest algorithm, such as SHA-256.

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
```

The program may optionally request an implementation from a specific provider, by indicating the provider name, as in the following:

```
MessageDigest md =
    MessageDigest.getInstance("SHA-256", "ProviderC");
```

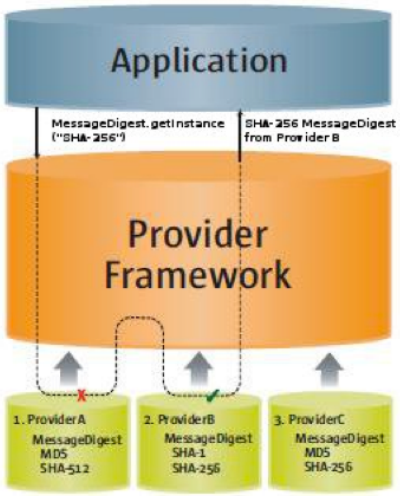


Figure 1 Provider searching

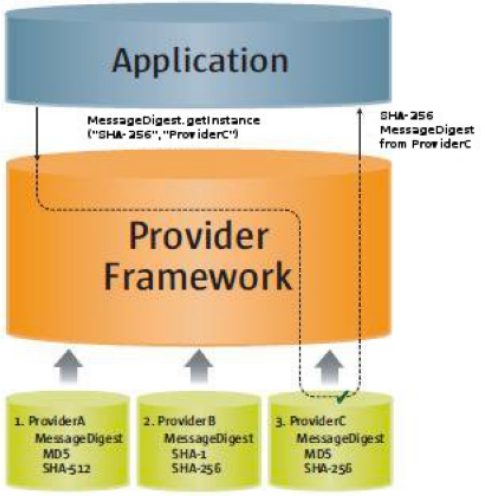


Figure 2 Specific provider requested

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

Cryptography Architecture

4 Cryptography

The Java cryptography architecture is a framework for accessing and developing cryptographic functionality for the Java platform. It includes APIs for a large variety of cryptographic services, including

- Message digest algorithms
- Digital signature algorithms
- Symmetric bulk encryption
- Symmetric stream encryption
- Asymmetric encryption
- Password-based encryption (PBE)
- Elliptic Curve Cryptography (ECC)
- Key agreement algorithms
- Key generators
- Message Authentication Codes (MACs)
- (Pseudo-)random number generators

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

PKI Support

5 Public Key Infrastructure

Public Key Infrastructure (PKI) is a term used for a framework that enables secure exchange of information based on public key cryptography. It allows identities (of people, organizations, etc.) to be bound to digital certificates and provides a means of verifying the authenticity of certificates. PKI encompasses keys, certificates, public key encryption, and trusted Certification Authorities (CAs) who generate and digitally sign certificates.

The Java platform includes APIs and provider support for X.509 digital certificates and Certificate Revocation Lists (CRLs), as well as PKIX-compliant certification path building and validation. The classes related to PKI are located in the `java.security` and `java.security.cert` packages.

Key and Certificate Storage

The Java platform provides for long-term persistent storage of cryptographic keys and certificates via key and certificate stores. Specifically, the `java.security.KeyStore` class represents a *key store*, a secure repository of cryptographic keys and/or trusted certificates (to be used, for example, during certification path validation), and the `java.security.cert.CertStore` class represents a *certificate store*, a public and potentially vast repository of unrelated and typically untrusted certificates. A `CertStore` may also store CRLs.

`KeyStore` and `CertStore` implementations are distinguished by types. The Java platform includes the standard *PKCS11* and *PKCS12* key store types (whose implementations are compliant with the corresponding PKCS specifications from RSA Security). It also contains a proprietary file-based key store type called *JKS* (which stands for "Java Key Store"), and a type called DKS ("Domain Key Store") which is a collection of keystores that are presented as a single logical keystore.

The Java platform includes a special built-in JKS key store, *cacerts*, that contains a number of certificates for well-known, trusted CAs. The keytool utility is able to list the certificates included in *cacerts* (see the security features documentation link in Section 10).

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

PKI Tools

PKI Tools

There are two built-in tools for working with keys, certificates, and key stores:

keytool is used to create and manage key stores. It can

- Create public/private key pairs
- Display, import, and export X.509 v1, v2, and v3 certificates stored as files
- Create self-signed certificates
- Issue certificate (PKCS#10) requests to be sent to CAs
- Create certificates based on certificate requests
- Import certificate replies (obtained from the CAs sent certificate requests)
- Designate public key certificates as trusted
- Accept a password and store it securely as a secret key

The **jarsigner** tool is used to sign JAR files, or to verify signatures on signed JAR files. The Java ARchive (JAR) file format enables the bundling of multiple files into a single file. Typically a JAR file contains the class files and auxiliary resources associated with applets and applications. When you want to digitally sign code, you first use keytool to generate or import appropriate keys and certificates into your key store (if they are not there already), then use the **jar** tool to place the code in a JAR file, and finally use the jarsigner tool to sign the JAR file. The jarsigner tool accesses a key store to find any keys and certificates needed to sign a JAR file or to verify the signature of a signed JAR file. Note: **jarsigner** can optionally generate signatures that include a timestamp. Systems (such as Java Plug-in) that verify JAR file signatures can check the timestamp and accept a JAR file that was signed while the signing certificate was valid rather than requiring the certificate to be current. (Certificates typically expire annually, and it is not reasonable to expect JAR file creators to re-sign deployed JAR files annually.)

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

Authentication

6 Authentication

Authentication is the process of determining the identity of a user. In the context of the Java runtime environment, it is the process of identifying the user of an executing Java program. In certain cases, this process may rely on the services described in the "Cryptography" section (Section 4).

The Java platform provides APIs that enable an application to perform user authentication via pluggable login modules. Applications call into the `LoginContext` class (in the `javax.security.auth.login` package), which in turn references a configuration. The configuration specifies which login module (an implementation of the `javax.security.auth.spi.LoginModule` interface) is to be used to perform the actual authentication.

Since applications solely talk to the standard `LoginContext` API, they can remain independent from the underlying plug-in modules. New or updated modules can be plugged in for an application without having to modify the application itself. Figure 3 illustrates the independence between applications and underlying login modules:

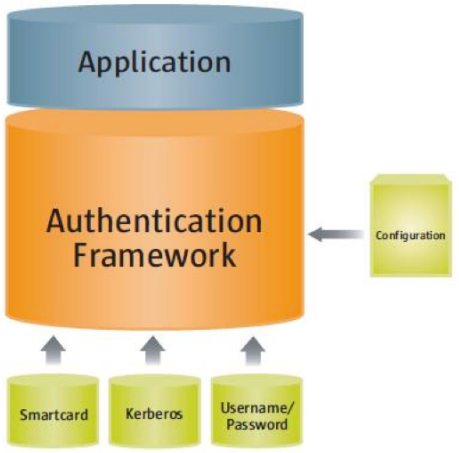


Figure 3 Authentication login modules plugging into the authentication framework

The Java platform provides the following built-in LoginModules, all in the `com.sun.security.auth.module` package:

- `Krb5LoginModule` for authentication using Kerberos protocols
- `JndiLoginModule` for username/password authentication using LDAP or NIS databases
- `KeyStoreLoginModule` for logging into any type of key store, including a PKCS#11 token key store

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

Access Control

8 Access Control

The access control architecture in the Java platform protects access to sensitive resources (for example, local files) or sensitive application code (for example, methods in a class). All access control decisions are mediated by a security manager, represented by the `java.lang.SecurityManager` class. A `SecurityManager` must be installed into the Java runtime in order to activate the access control checks.

Java applets and Java™ Web Start applications are automatically run with a `SecurityManager` installed. However, local applications executed via the **java** command are by default not run with a `SecurityManager` installed. In order to run local applications with a SecurityManager, either the application itself must programmatically set one via the `setSecurityManager` method (in the `java.lang.System` class), or **java** must be invoked with a `-Djava.security.manager` argument on the commandline.

Permissions

When Java code is loaded by a class loader into the Java runtime, the class loader automatically associates the following information with that code:

- Where the code was loaded from
- Who signed the code (if anyone)
- Default permissions granted to the code

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

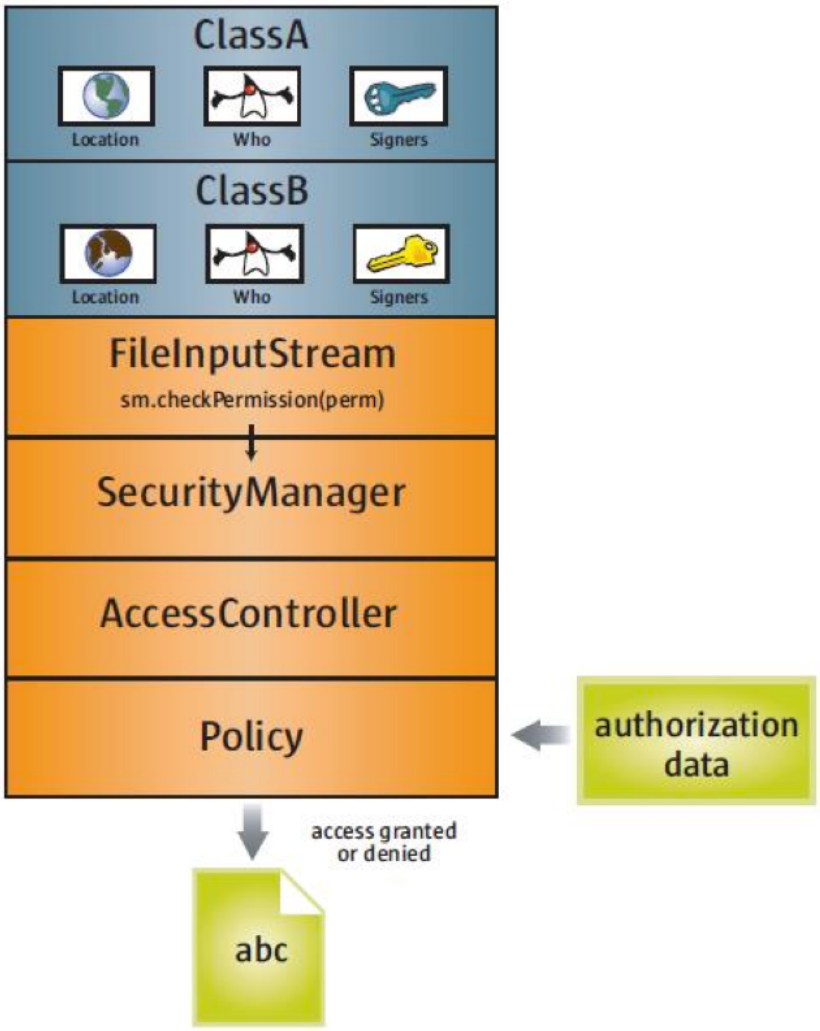


Figure 4 Controlling access to resources

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

9 XML Signature

The Java XML Digital Signature API is a standard Java API for generating and validating XML Signatures.

XML Signatures can be applied to data of any type, XML or binary (see http://www.w3.org/TR/xmldsig-core/). The resulting signature is represented in XML. An XML Signature can be used to secure your data and provide data integrity, message authentication, and signer authentication.

The API is designed to support all of the required or recommended features of the W3C Recommendation for XML-Signature Syntax and Processing. The API is extensible and pluggable and is based on the Java Cryptography Service Provider Architecture.

The Java XML Digital Signature APIs consist of six packages:

- `javax.xml.crypto`
- `javax.xml.crypto.dsig`
- `javax.xml.crypto.dsig.keyinfo`
- `javax.xml.crypto.dsig.spec`
- `javax.xml.crypto.dom`
- `javax.xml.crypto.dsig.dom`

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

Appendix A Classes Summary

Table 1 summarizes the names, packages, and usage of the Java security classes and interfaces mentioned in this paper.

Table 1 Key Java security packages and classes

Package	Class/Interface Name	Usage
com.sun.security.auth.module	JndiLoginModule	Performs username/password authentication using LDAP or NIS
com.sun.security.auth.module	KeyStoreLoginModule	Performs authentication based on key store login
com.sun.security.auth.module	Krb5LoginModule	Performs authentication using Kerberos protocols
java.lang	SecurityException	Indicates a security violation
java.lang	SecurityManager	Mediates all access control decisions
java.lang	System	Installs the SecurityManager
java.security	AccessController	Called by default implementation of SecurityManager to make access control decisions
java.security	DomainLoadStoreParameter	Stores parameters for the Domain keystore (DKS)
java.security	Key	Represents a cryptographic key
java.security	KeyStore	Represents a repository of keys and trusted certificates
java.security	MessageDigest	Represents a message digest
java.security	Permission	Represents access to a particular resource
java.security	PKCS12Attribute	Supports attributes in PKCS12 keystores
java.security	Policy	Encapsulates the security policy
java.security	Provider	Encapsulates security service implementations
java.security	Security	Manages security providers and security properties
java.security	Signature	Creates and verifies digital signatures

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

java.security.cert	Certificate	Represents a public key certificate
java.security.cert	CertStore	Represents a repository of unrelated and typically untrusted certificates
java.security.cert	CRL	Represents a CRL
javax.crypto	Cipher	Performs encryption and decryption
javax.crypto	KeyAgreement	Performs a key exchange
javax.net.ssl	KeyManager	Manages keys used to perform SSL/TLS authentication
javax.net.ssl	SSLEngine	Produces/consumes SSL/TLS packets, allowing the application freedom to choose a transport mechanism
javax.net.ssl	SSLSocket	Represents a network socket that encapsulates SSL/TLS support on top of a normal stream socket
javax.net.ssl	TrustManager	Makes decisions about who to trust in SSL/TLS interactions (for example, based on trusted certificates in key stores)
javax.security.auth	Subject	Represents a user
javax.security.auth.kerberos	KerberosPrincipal	Represents a Kerberos principal
javax.security.auth.kerberos	KerberosTicket	Represents a Kerberos ticket
javax.security.auth.kerberos	KerberosKey	Represents a Kerberos key
javax.security.auth.kerberos	KerberosTab	Represents a Kerberos keytab file
javax.security.auth.login	LoginContext	Supports pluggable authentication
javax.security.auth.spi	LoginModule	Implements a specific authentication mechanism
javax.security.sasl	Sasl	Creates SaslClient and SaslServer objects
javax.security.sasl	SaslClient	Performs SASL authentication as a client
javax.security.sasl	SaslServer	Performs SASL authentication as a server
org.ietf.jgss	GSSContext	Encapsulates a GSS-API security context and provides the security services available via the context

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

Appendix B Tools Summary

Table 2 summarizes the tools mentioned in this paper.

Table 2 Java security tools

Tool	Usage
`jar`	Creates Java Archive (JAR) files
`jarsigner`	Signs and verifies signatures on JAR files
`keytool`	Creates and manages key stores
`policytool`	Creates and edits policy files for use with default Policy implementation

There are also three Kerberos-related tools that are shipped with the Java platform for Windows. Equivalent functionality is provided in tools of the same name that are automatically part of the Solaris and Linux operating environments. Table 3 summarizes the Kerberos tools.

Table 3 Kerberos-related tools

Tool	Usage
`kinit`	Obtains and caches Kerberos ticket-granting tickets
`klist`	Lists entries in the local Kerberos credentials cache and key table
`ktab`	Manages the names and service keys stored in the local Kerberos key table

Appendix C Built-in Providers

The Java platform implementation from Oracle includes a number of built-in provider packages. For details, see the Java™ Cryptography Architecture Oracle Providers Documentation.

http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html

Trail: Security Features in Java SE

Trail Lessons

Creating a Policy File shows how resource accesses can be controlled by a policy file. For latest information on policy configuration files, see Policy Guide page.

Quick Tour of Controlling Applications builds on the previous lesson, showing how resource accesses, such as reading or writing a file, are not permitted for applications that are run under a security manager unless explicitly allowed by a permission in a policy file.

API and Tools Use for Secure Code and File Exchanges defines digital signatures, certificates, and keystores and discusses why they are needed. It also reviews information applicable to the next three lessons regarding the steps commonly needed for using the tools or the API to generate signatures, export/import certificates, and so on.

Signing Code and Granting It Permissions illustrates the use of all the security-related tools. It shows the steps that a developer would take to sign and to distribute code for others to run. The lesson also shows how someone who will run the code (or a system administrator) could add an entry in a policy file to grant the code permission for the resource accesses it needs.

Exchanging Files shows use of the tools by one person to sign an important document, such as a contract, and to export the public key certificate for the public key corresponding to the private key used to sign the contract. Then the lesson shows how another person, who receives the contract, the signature, and the public key certificate, can import the certificate and verify the signature.

Generating and Verifying Signatures walks you step by step through an example of writing a Java program using the JDK Security API to generate keys, to generate a digital signature for data using the private key, and to export the public key and the signature to files. Then the example shows writing a second program, which may be expected to run on a different person's computer, that imports the public key and verifies the authenticity of the signature. Finally, the example discusses potential weaknesses of the approach used by the basic programs and demonstrates possible alternative approaches and methods of supplying and importing keys, including in certificates.

Implementing Your Own Permission demonstrates how to write a class that defines its own special permission.

http://docs.oracle.com/javase/tutorial/security/index.html

Policy Tool

Set up a Policy File to Grant the Required Permission

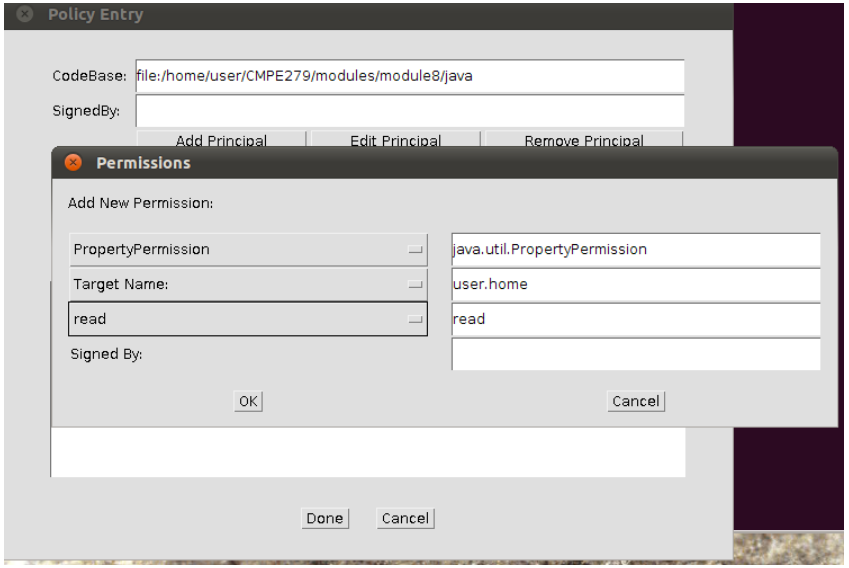
A policy file is an ASCII text file and can be composed via a text editor or the graphical Policy Tool utility demonstrated in this section. The Policy Tool saves you typing and eliminates the need for you to know the required syntax of policy files, thus reducing errors.

This lesson uses the Policy Tool to create a policy file named **`examplepolicy`**, in which you will add a *policy entry* that grants code from the **`examples`** directory permission to write.

Follow these steps to create and modify your new policy file:

1. Start Policy Tool
2. Grant the Required Permission
3. Save the Policy File

Note for UNIX Users: The steps illustrate creating the policy file for a Windows system. The steps are exactly the same if you are working on a UNIX system. Where the text says to store the policy file in the `C:\Test` directory, you can store it in another directory. The examples in the lesson Quick Tour of Controlling Applications assume that you stored it in the `~/test` directory.



Application Access Control

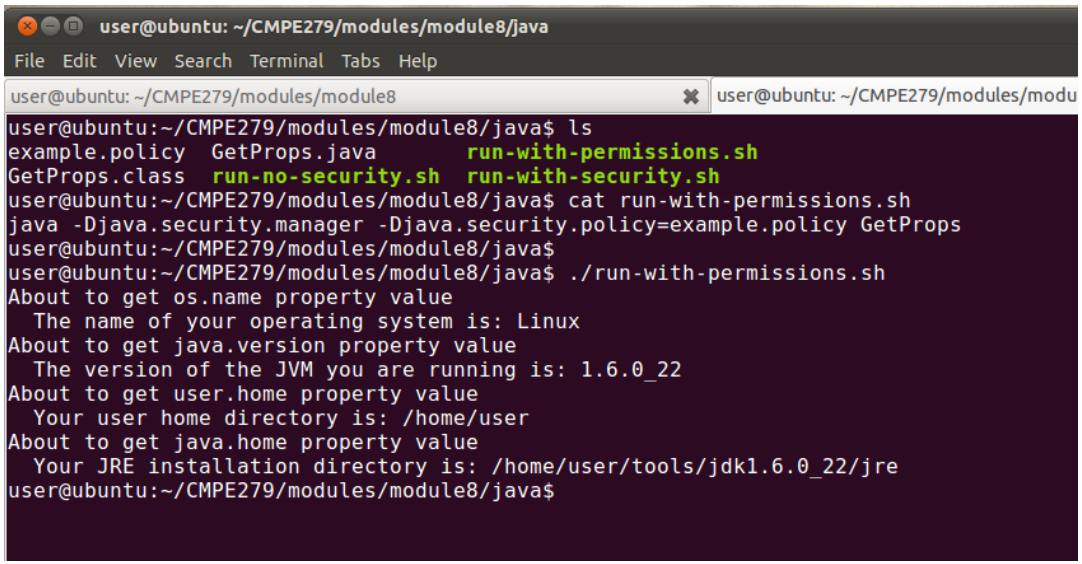
Lesson: Quick Tour of Controlling Applications

Pre-requisite lesson: Creating a Policy File

This lesson shows how to use a security manager to grant or deny access to system resources for applications. The lesson also shows how resource accesses, such as reading or writing a file, are not permitted for applications that are run with a security manager unless explicitly allowed by a permission entry in a policy file.

The steps for this lesson are:

1. Observe Application Freedom
2. See How to Restrict Applications
3. Set Up the Policy File to Grant the Required Permissions
4. See the Policy File Effects



Tools for Secure Code & File Exchanges

Lesson: API and Tools Use for Secure Code and File Exchanges

This lesson explains why digital signatures, certificates, and keystores are needed. The lesson also compares use of the tools versus the JDK Security API with respect to generating signatures. Such tool usage is demonstrated in the next two lessons, Signing Code and Granting It Permissions and Exchanging Files. API usage is demonstrated in the Generating and Verifying Signatures lesson.

This lesson contains the following sections

- Code and Document Security
- Tool and API Notes
- Use of the JDK Security API to Sign Documents
- Use of the Tools to Sign Code or Documents

JAR Signing

Lesson: Signing Code and Granting It Permissions

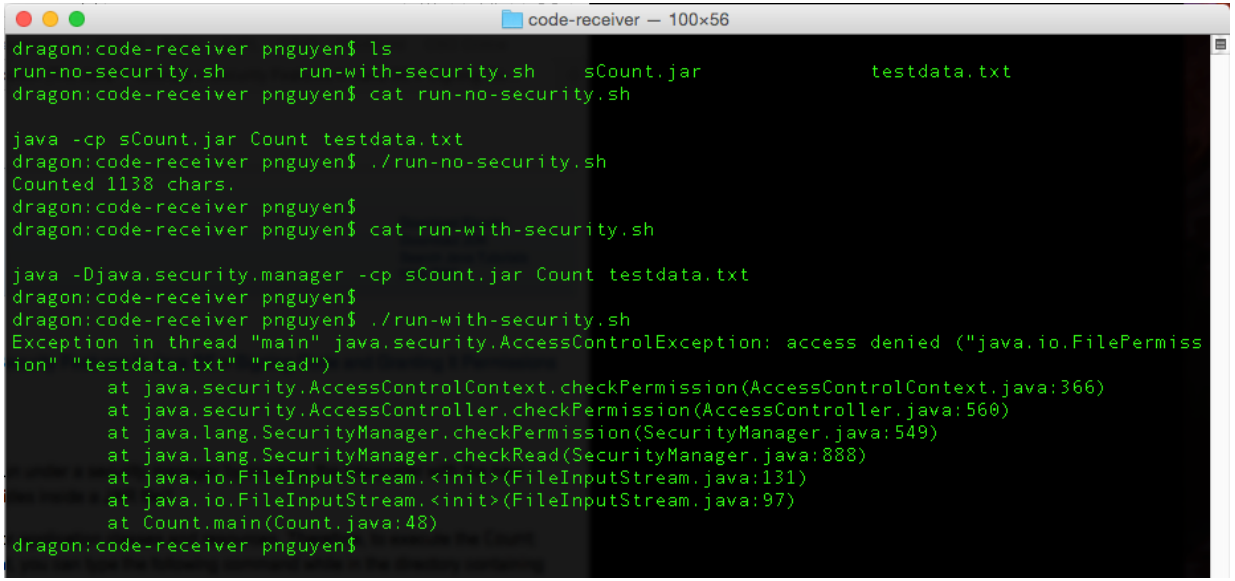
This lesson shows how to use use `keytool`, `jarsigner`, `Policy Tool` and `jar` to place files into JAR (Java ARchive) files for subsequent signing by the `jarsigner` tool.

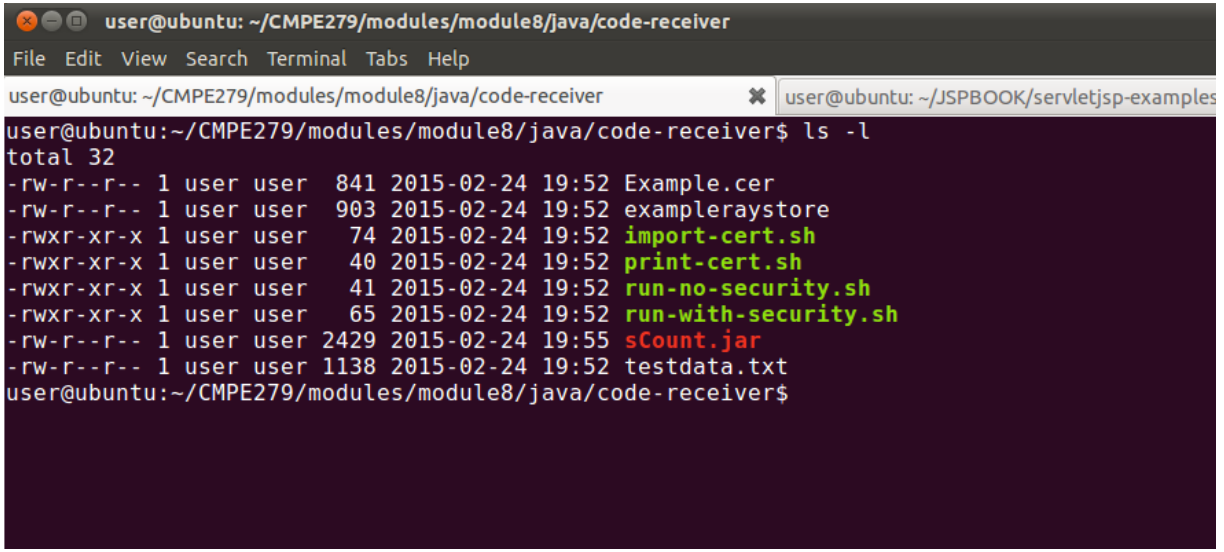
This lesson has two parts. First, you will create and deploy an application. Second; you will act as the recipient of a signed application.

Here are the steps to create and deploy an application:

Note: For convenience, you pretend to be a user/developer named Susan Jones. You need to define Susan Jones when you generate the keys.

- Put Java class files comprising your application into a JAR file
- Sign the JAR file
- Export the public key certificate corresponding to the private key used to sign the JAR file





Signed Documents

Lesson: Exchanging Files

If you want to electronically send an important document, (like a Contract) to someone else, it is a good idea to digitally "sign" the document, so your recipient can check that the document indeed came from you and was not altered in transit.

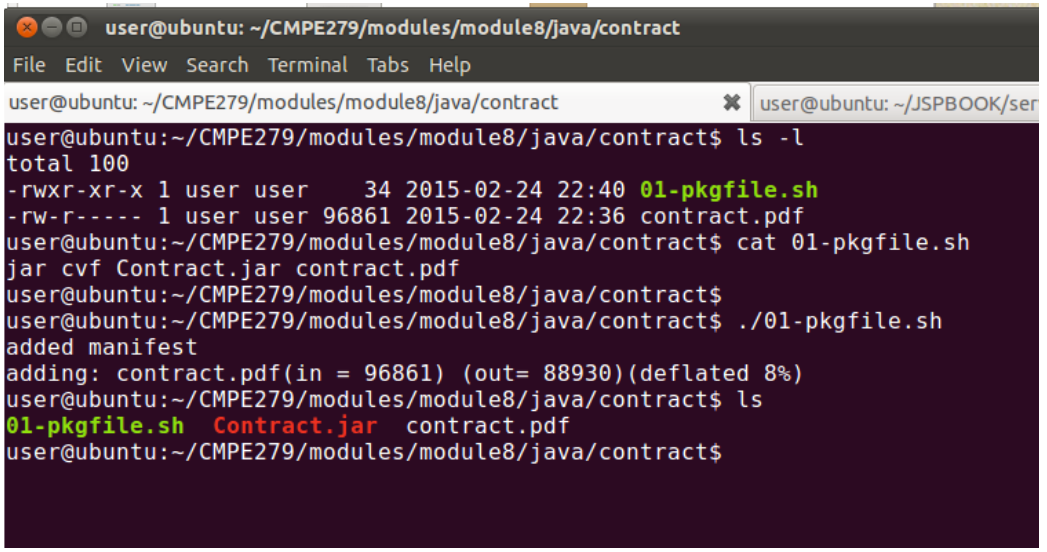
This lesson shows you how to use Security tools for the exchange of an important document, in this case a contract.

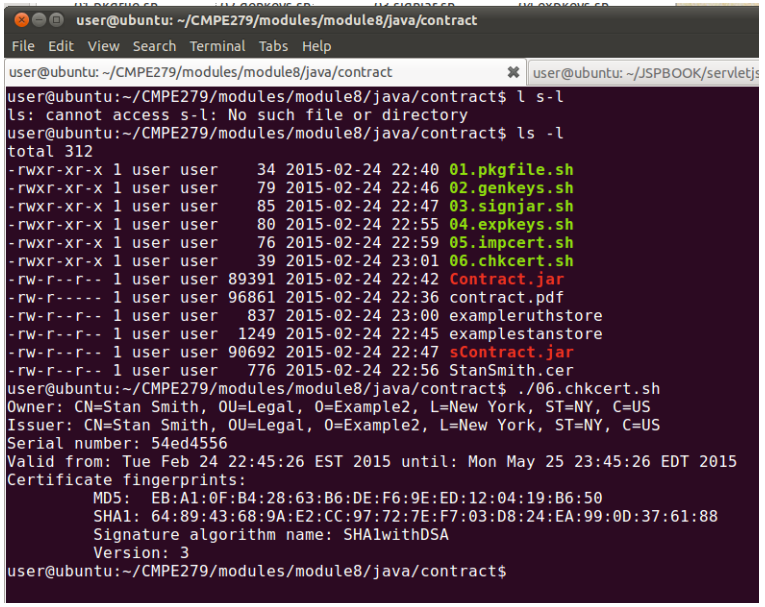
You first pretend that you are the contract sender, Stan Smith. This lesson shows the steps Stan would use to put the contract in a JAR file, sign it, and export the public key certificate for the public key corresponding to the private key used to sign the JAR file.

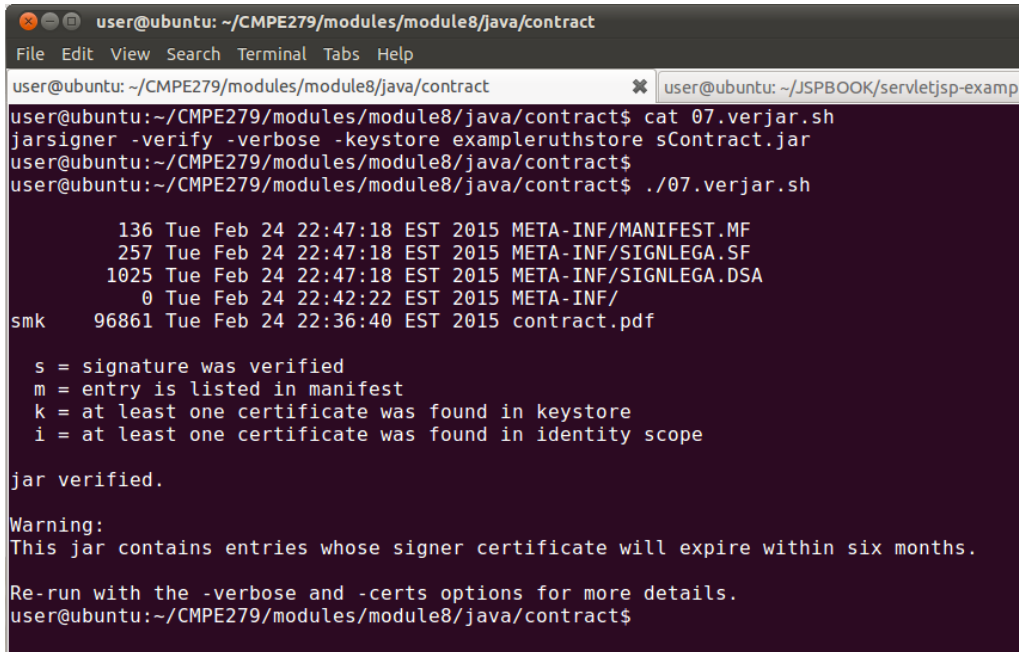
Then you pretend that you are Ruth, who has received the signed JAR file and the certificate. You'll use `keytool` to import the certificate into Ruth's keystore in an entry aliased by `stan`, and use the `jarsigner` tool to verify the signature.

For further information about digital signatures, certificates, keystores, and the tools, see the API and Tools Use for Secure Code and File Exchanges lesson.

Note: This lesson assumes that you execute all commands from within the same directory.







Signature Verification

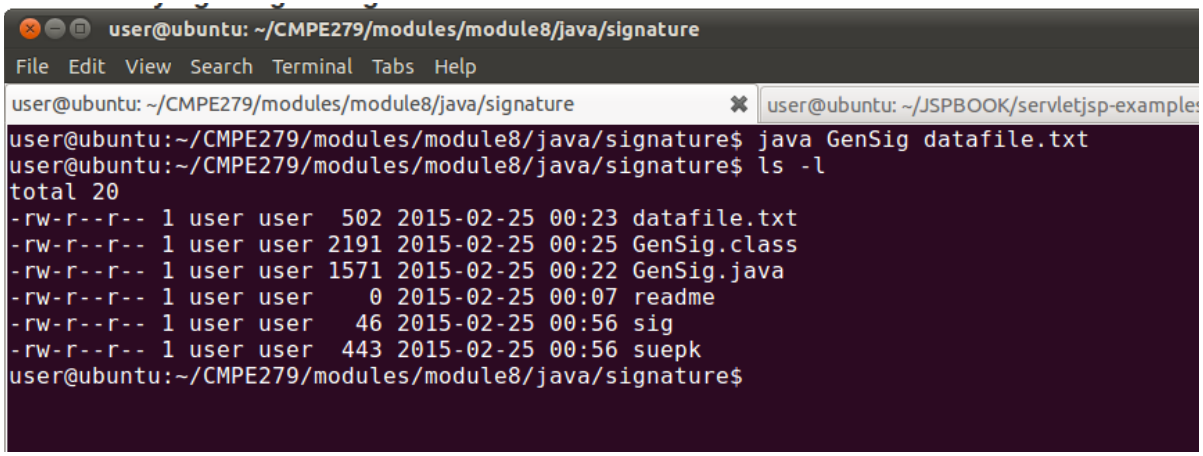
Lesson: Generating and Verifying Signatures

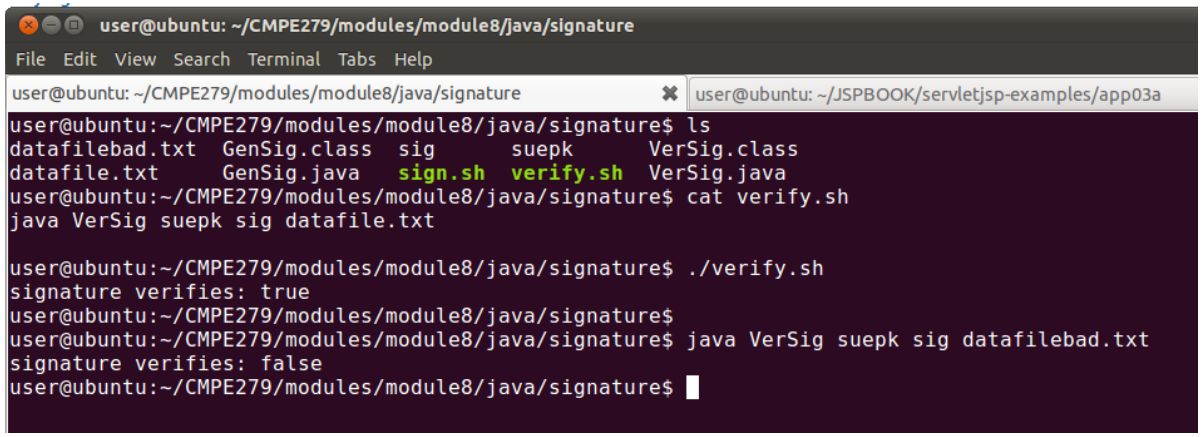
This lesson walks you through the steps necessary to use the JDK Security API to generate a digital signature for data and to verify that a signature is authentic. This lesson is meant for developers who wish to incorporate security functionality into their programs, including cryptography services.

This lesson demonstrates the use of the JDK Security API with respect to signing documents. The lesson shows what one program, executed by the person who has the original document, would do to generate keys, generate a digital signature for the document using the private key, and export the public key and the signature to files.

Then it shows an example of another program, executed by the receiver of the document, signature, and public key. It shows how the program could import the public key and verify the authenticity of the signature. The lesson also discusses and demonstrates possible alternative approaches and methods of supplying and importing keys, including in certificates.

For further information about the concepts and terminology (digital signatures, certificates, keystores), see the API and Tools Use for Secure Code and File Exchanges lesson.





Custom Permissions

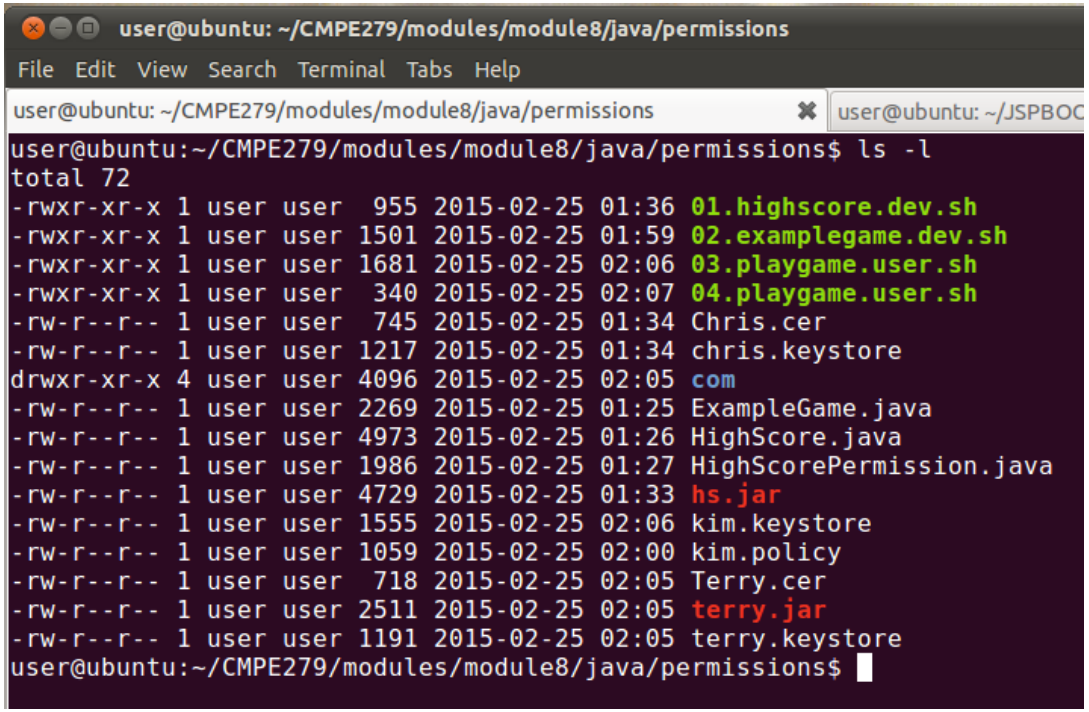
Lesson: Implementing Your Own Permission

This lesson demonstrates how to write a class that defines its own special permission. The basic components in this lesson include:

1. A sample game called **ExampleGame**.
2. A class called **HighScore**, which is used by `ExampleGame` to store a user's latest high score.
3. A class called **HighScorePermission**, which is used to protect access to the user's stored high score value.
4. A user's security **policy file**, which grants permission to `ExampleGame` to update his/her high score.

The basic scenario is as follows:

1. A user plays `ExampleGame`.
2. If the user reaches a new high score, `ExampleGame` uses the `HighScore` class to save this new value.
3. The `HighScore` class looks into the user's security policy to check if `ExampleGame` has permission to update the user's high score value.
4. If `ExampleGame` has permission to update the high score, then the HighScore class updates that value.



OWASP Exploits

To be Continued…

Next: OWASP and WebGoat Hacking

- *Security Compass OWASP Top 10 Videos*
- *WebGoat Hacking Training Videos*

2015.03.04