

LS/EECS

Building E-commerce Applications LAB Setup

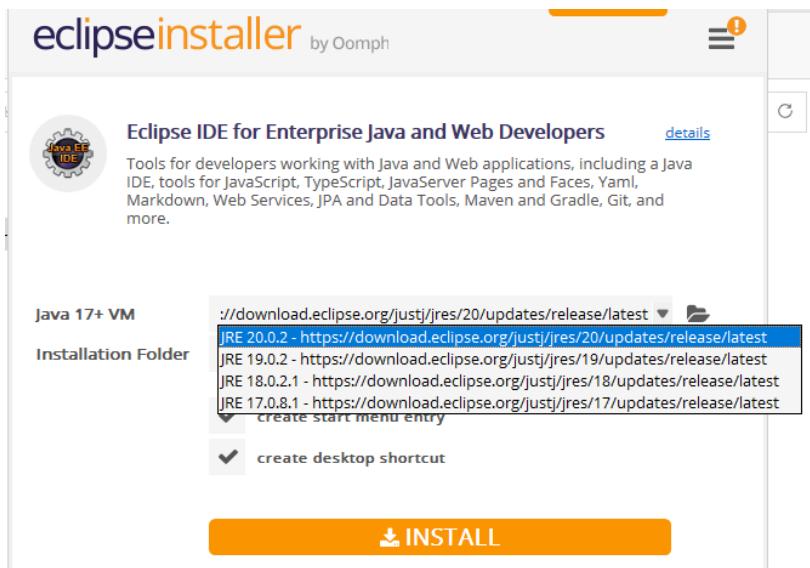
Objectives

- understand the development environment
- Setup the development environment
- explore the servlet programming and servlet APIs

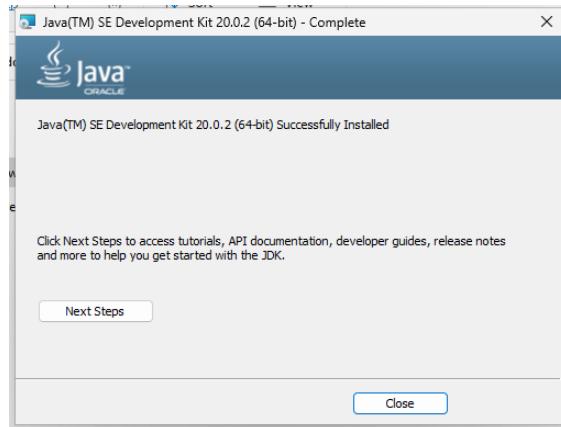
Task A: Setup and test the Dev and Runtime Environment

(on Lab computers, skip 1-3)

1. Download and install Eclipse “Eclipse IDE for Java Enterprise Developers,” from [eclipse.org](https://www.eclipse.org/downloads/packages/), <https://www.eclipse.org/downloads/packages/>. Use the latest stable version.



- a. Need to install Java JDK on your computer
<https://www.oracle.com/ca-en/java/technologies/downloads/>
- b. Use installers, based on your operating system and processor.



- c. Start Eclipse
- d. Explore the Perspectives
- e. Run your first Java program, Hello.java

2. Install Apache Tomcat (v10), <https://tomcat.apache.org/download-10.cgi>

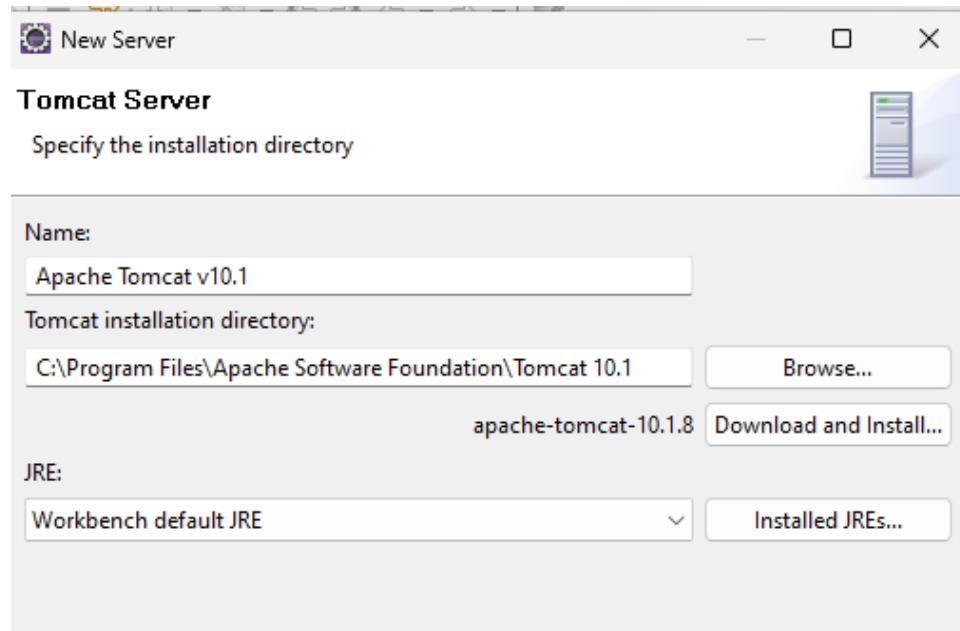
For Windows, you can use the 32-bit/64-bit Windows Service Installer

For Unix-based,

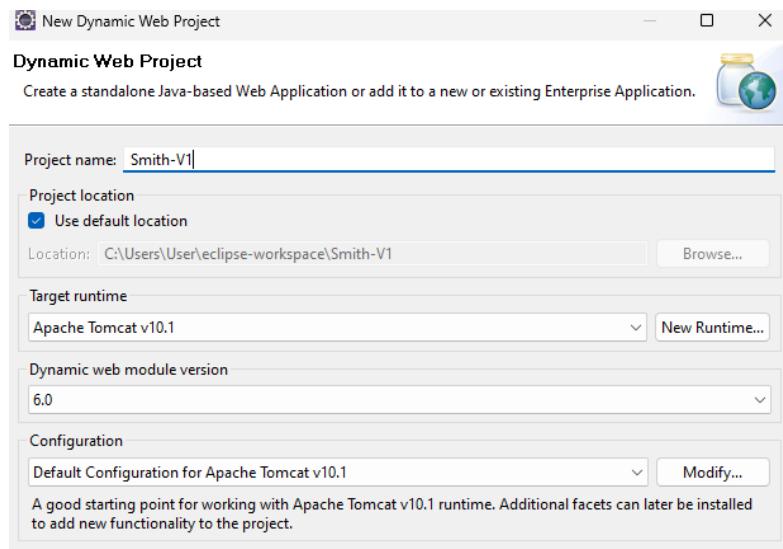
- a. Download Tomcat 10 Core tar.gz file
- b. Open Terminal and unarchive file
- C. sudo mkdir -p /usr/local
- d. sudo mv ~/Downloads/apache-tomcat-10.1.8 /usr/local
- e. Run startup.sh script Tomcat should be running on <http://localhost:8080>

The screenshot shows the Apache Tomcat 10.1.13 home page. At the top, there is a navigation bar with links to Home, Documentation, Configuration, Examples, Wiki, and Mailing Lists. On the right, there is a "Find Help" search bar and the Apache Software Foundation logo. The main content area features a green banner with the text "If you're seeing this, you've successfully installed Tomcat. Congratulations!". Below the banner is a cartoon cat icon and a section titled "Recommended Reading" with links to Security Considerations How-To, Manager Application How-To, and Clustering/Session Replication How-To. To the right of the banner are three buttons: Server Status, Manager App, and Host Manager. The page is divided into several sections: "Developer Quick Start" (with links to Tomcat Setup, First Web Application, Realms & AAA, JDBC DataSources, Examples, Servlet Specifications, and Tomcat Versions); "Managing Tomcat" (with information about manager webapp access and a link to Read more...); "Documentation" (with links to Tomcat 10.1 Documentation, Tomcat 10.1 Configuration, and Tomcat Wiki); and "Getting Help" (with a list of mailing lists: tomcat-announce, tomcat-users, and taglibs-user).

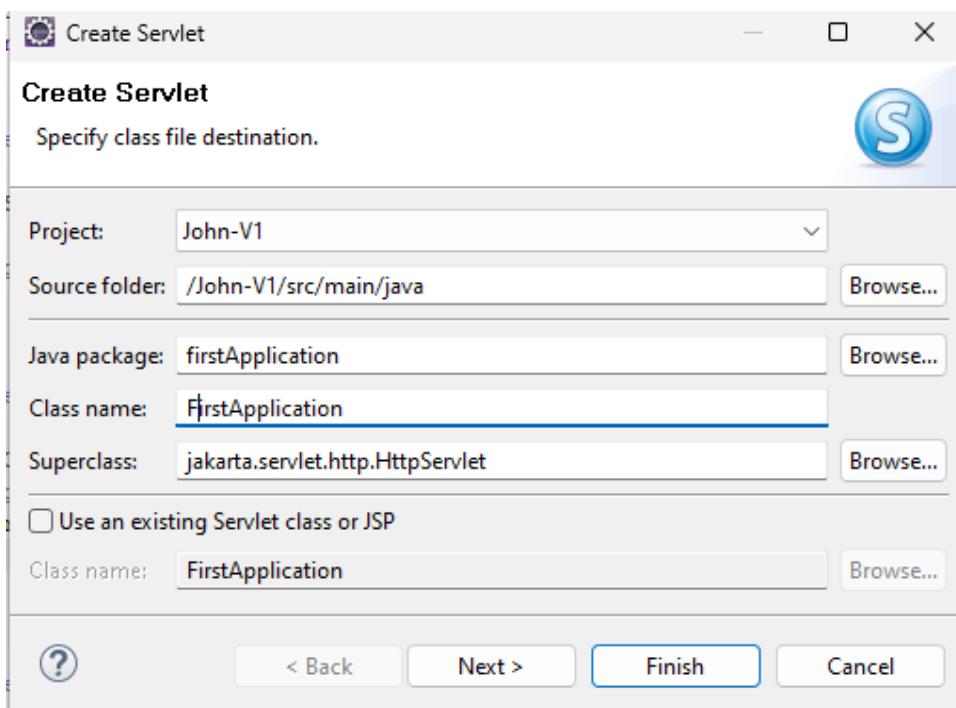
3. Configure Eclipse with Tomcat. In Java EE perspective, explore the Servers tab;
 - a. Create a Web server, select Apache Tomcat 10.0 (point to the directory where you unzipped Tomcat)



- b. Make sure to stop the Apache service when you first installed it, and run the **Tomcat v10.1** Server from Eclipse
4. Create your first Web Dynamic Project, LastName-v1
 - a. The project should have a `web.xml` file (last page of the create wizard)
 - b. The project name should start with your last name
 - c. Understand the structure of a Java EE project



5. Create a Java Servlet



6. Explore and play around with the Java Servlet Application

- Add `System.out.println("Hello, Got a GET request from the First Application!");` to the `DoGet()` function

To reply to the client (browser) with a message, in the servlet `FirstApplication.java`, `doGet()` method, you need to obtain a writer from response object and write a string.
`PrintWriter resOut = response.getWriter();`

```
resOut.write("Hello, 4413!\n");
```

- b. Run Application on the Server
- c. <http://localhost:8080/<LastName>-V1/FirstApplication>
On the Java Eclipse Console, you will see your Hello World command up
- d. Create new HTML File on your dynamic Application, name it HelloWorld
You can access it on <http://localhost:8080/Smith-V1>HelloWorld.html>

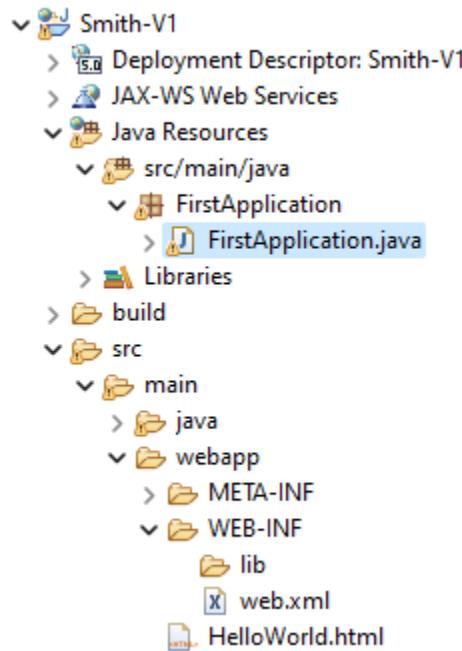


Fig. 1 A typical Java EE project in Eclipse: `src` folder contains the java files, including servlets; `webapp` folder contains the html pages. EE stands for Enterprise Edition

Task B: Explore the servlet, request and response API

1. Modify your servlet to respond to `/FirstApplication/*` requests in addition to `/FirstApplication`.
 - a. You can add additional paths to the annotation on top of the Servlet class:
`@WebServlet({ "/FirstApplication", "/FirstApplication/*" })`
 - b. Select the servlet, and then “Run on Server” from pop-up menu
 - c. Note that with the annotation “`/FirstApplication/*`” you can add any path and query to your invocation. Try it!
 - d. **Note: Some versions of Tomcat does not allow Servlet Mapping on both the Java File and Web.xml, If you get errors, go to the web.xml, and change the url-pattern instead**

Node	Content
?? xml	version="1.0" encoding="UTF-8"
web-app	(module-name? (((description*, display-name*, icon*) distributable context-param filter filter-mappi...
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xmlns	https://jakarta.ee/xml/ns/jakartae
xsi:schemaLocation	https://jakarta.ee/xml/ns/jakartae https://jakarta.ee/xml/ns/jakartae/web-app_6_0.xsd
version	6.0
servlet	((description*, display-name*, icon*), servlet-name, (servlet-class jsp-file?), init-param*, load-on-startup?,...
servlet-mapping	(servlet-name, url-pattern+)
servlet-name	FirstApplication
url-pattern	/FirstApplication/*
display-name	John-VI
welcome-file-list	(welcome-file+)

e.

2. After your application ‘Smith-v1’ is deployed, you can access its services with “curl”, from a terminal window. If you do not have curl on your computer, download if from <https://curl.se/>. Curl helps send http messages from CLI.
`~curl -X GET 'http://localhost:8080/Smith-V1/FirstApplication'`
3. By exploring the API of the servlet, add code to FirstApplication.java to do the following:
 - a. Return the client IP and client port to the browser
 How would you implement an IP-filtering firewall?
 - b. Return the request's http protocol and method
 - c. Return the request path
 - d. Pass a query string in the URL and return the entire query string sent by the client:
 - i. Use the `getParameter` on request object to extract a named request parameter
 - ii. Return the value of a parameter “foo”
 - e. Embed spaces in the query string and note the URL encoding in the developer tool.

NOTES ON HOW TO DO IT:

To explore different elements of the http request message, use get methods on request object...

```
String clientIP = request.getRemoteAddr();
resOut.write("Client IP: " + clientIP + "\n");
String clientQueryString = request.getQueryString();
String foo = request.getParameter("foo");
resOut.write("Query Param foo=" + foo + "\n");
```

At this point, you servlet response should look like in the caption below. Again, you can test it with curl or in the browser (note that any url in the browser url text box will invoke the GET method on the server.

```
Hello, World!
Client IP: 0:0:0:0:0:0:0:1
Client Port: 50076
This IP has been flagged!
Client Protocol: HTTP/1.1
Client Method: GET
Query String: foo=bar
Query Param foo=bar
Request URI : /OsapCalc-v1/Osap/aSamplePath
Request Servlet Path : /aSamplePath
```

Task C: Understanding Java EE Application Descriptor and ServletContext object

1. Deployment Descriptor consists of the file web.xml. The content of this file is accessible from your application, through the `ServletContext` class instances.

In this task, you:

- a. Add `FirstApplication` to the welcome file list and test it.
- b. Add a context parameter and verify that you can access it in the servlet.
- c. Explore the APIs of `ServletContext` class
- d. Add an error-page for error-code 404 and point it at location `/res/my404.html`. Test by visiting a non-existent page.
- e. Add an error-page for exception-type `java.lang.Exception` and point it at location `/res/myException.html`. Test by triggering any exception in your servlet.

NOTES ON HOW TO DO IT:

You should have a web.xml file in your project.

Edit web.xml using the xml editor that opens the file

- a. add `FirstApplication` to the file list, e.g.

```
<welcome-file-list>
    <welcome-file>FirstApplication</welcome-file>
    <welcome-file>index.html</welcome-file>
    ...

```

- b. Add context parameter children. The parameters have name and value

Ex:

```
<context-param>
    <param-name>applicationName</param-name>
    <param-value>My First Application</param-value>
</context-param>
<context-param>
    <param-name>applicantName</param-name>
    <param-value>John Smith</param-value>
</context-param>
```

NOTE:

web.xml is loaded when the application is deployed, you might need to restart the server after you edit it.

you can access the `ServletContext` from `doGet` with

```
ServletContext context= this.getServletContext();
```

You can read the context parameters from within the servlet, using the method

```
getServletContext().getInitParameter("parameterName")
//it returns a String.
```

Retrieve the following parameters and return them to client

- ApplicantName
- ApplicationName

Explore the following methods of the `ServletContext`: `getContextPath()` and `getRealPath()`. To better understand them, check what they return as in the example below

```
String contextPath=context.getContextPath();
String realPath=context.getRealPath("FirstApplication");
```

```
Hello, World!
Client IP: 0:0:0:0:0:0:1
Client Port: 50076
This IP has been flagged!
Client Protocol: HTTP/1.1
Client Method: GET
Query String: foo=bar
Query Param foo=bar
Request URI: /Smith-V1/FirstApplication/aSamplePath
Request Servlet Path: /aSamplePath
---- Infor from context object ---
Application Name: My First Application
Context Path: /Smith-V1
Real Path of FirstApplication Servlet: /Users/<username>/Documents/workspace/.metadata/.plugins/
org.eclipse.wst.server.core/tmp0/wtpwebapps/Smith-V0/FirstApplication
Applicant name: John Smith
```

Fig. 2. Servlet response with context info.

2. Error Pages

- you have to create them in webapp/res folder in your project
- you have to edit the web.xml and add error page children, like these:

```
<error-page>
    <error-code>404</error-code>
    <location>/res/my404.html</location>
</error-page>
<error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/res/myException.html</location>
</error-page>
```

- then create the pages. The pages should display some meaningful content when triggered.

Run Curl Commands to test your web application

- Hello World
`curl "http://localhost:8080/Smith-V1>HelloWorld.html"`
- Get Request
`curl "http://localhost:8080/<Lastname>-V1/FirstApplication"`
- Post Request:
`curl -d "" -X POST "http://localhost:8080/<LastName>-
V1/FirstApplication"`

- **Query Strings:**

```
curl "http://localhost:8080/<Lastname>-V1/FirstApplication?foo=bar"
```