Servlets

Programming with Web Technologies

Auckland
ICT Graduate School

What is a Web Server

A web server is a computer system that processes requests made from web browsers and sends responses back

A web server can serve either static or dynamic content

Static content refers to files that are stored on the web server that are served 'as-is' - these are the sorts of files we have been writing so far in the course

Dynamic content is a new concept for us, but refers to files that can be partly or completely generated when a request is received, allowing for custom pages

What is a Web Server

1. User logs into Facebook using a web browser

2. The URL specifies the server to connect to and page to request

3. The server receives the request and processes it. It then retrieves the requested page

4. The server sends the users personal Facebook page

5. The browser receives the message from the server and lays it out on the screen

What is a Web Server

Depending on the request, the server may carry out other operations. These could be retrieving information from a database or file, retrieving stylesheets, JavaScript, images or video stored on the same server, or from others.

What is a Web Server

Only clients can make HTTP requests

HTTP requests go to the server. The web server must answer every HTTP request

The web server is responsible for processing and answering all requests

The web server will:

1. Check if the requested URL matches an existing file
2. If the file exists, the server sends the file content back to the client, or generates the file
3. If no match is found, the web browser returns an error message (404)

Web Servers

There are many different web server implementations in use. Some of the more popular or well known ones are

- Apache HTTP Server
- Internet Information Server (IIS)
- Nginx
- lighttpd
- Apache Tomcat
- Jetty
- Node.js

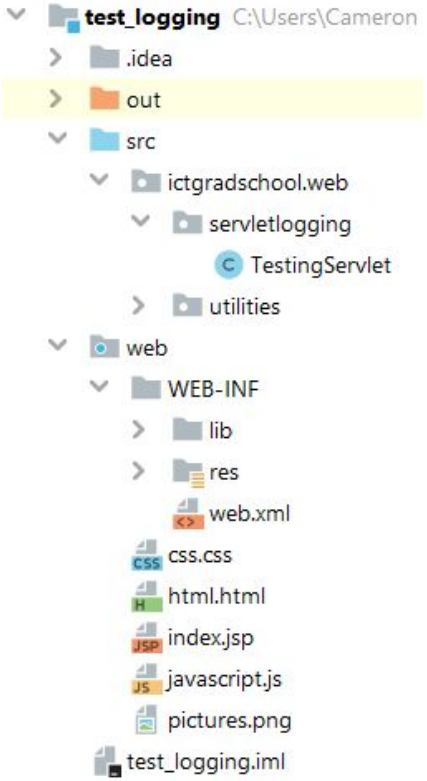
Definition: Java Servlet

"A servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol" [1]

"A Java servlet is a Java program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers. Such Web servlets are the Java counterpart to other dynamic Web content technologies such as PHP"[2]

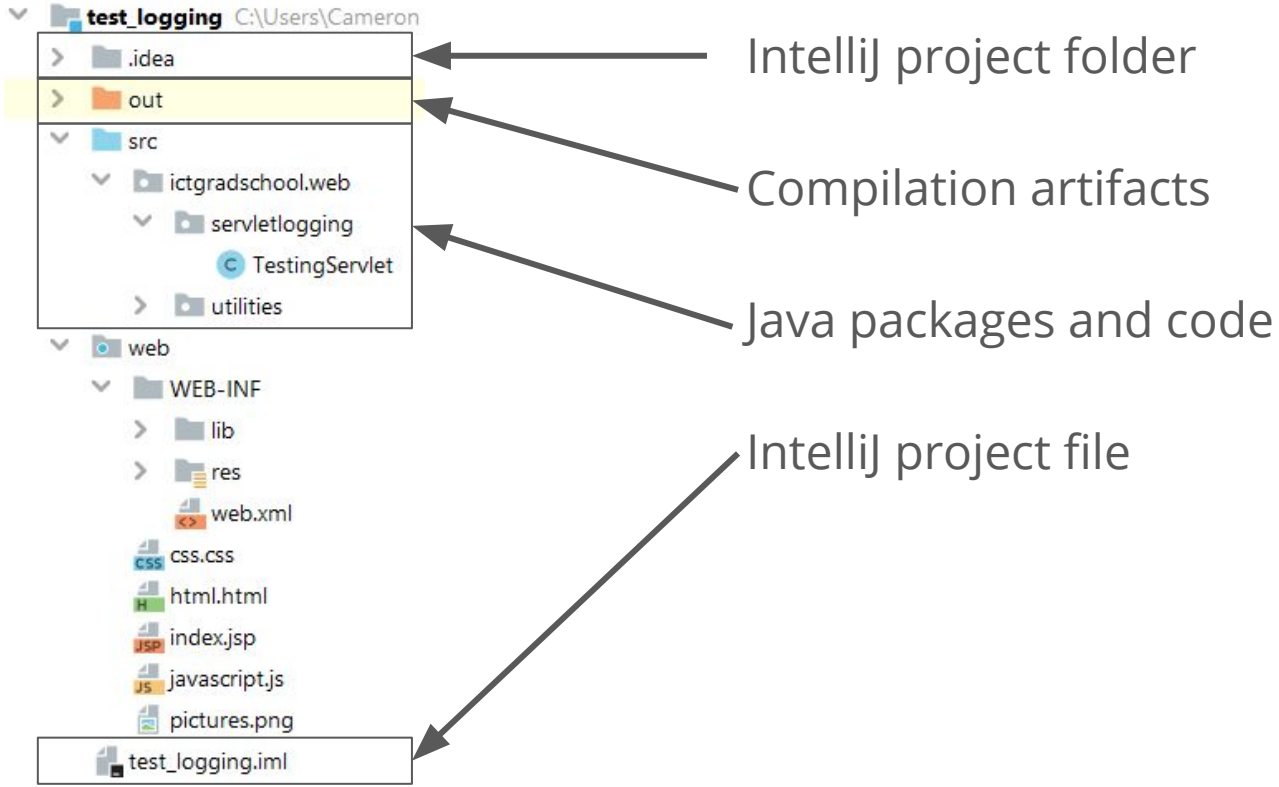
We will only be developing Web/HTTP Servlets

Servlet Directory Structure

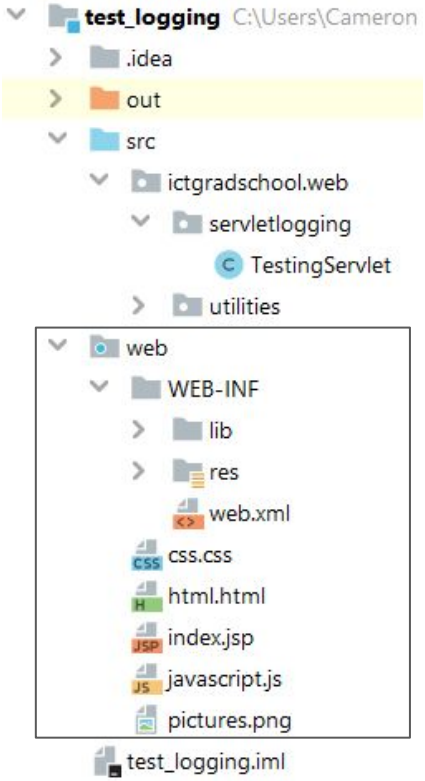


Working with a servlet project is going to feel quite familiar - it looks a lot like a regular Java project

Servlet Directory Structure

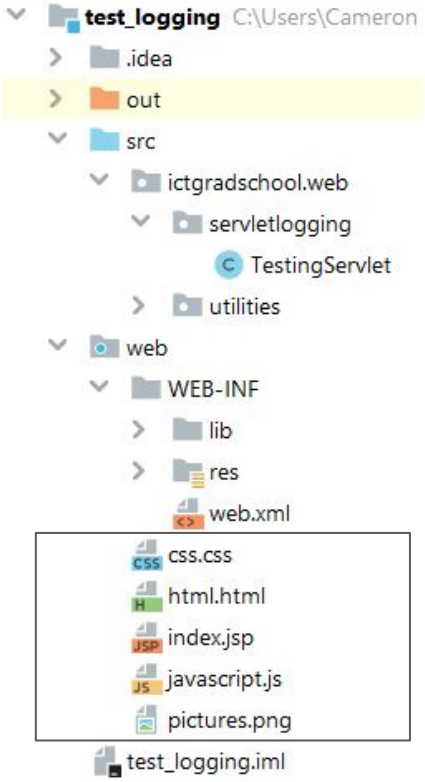


Servlet Directory Structure



The **web** folder though, is new

Servlet Directory Structure

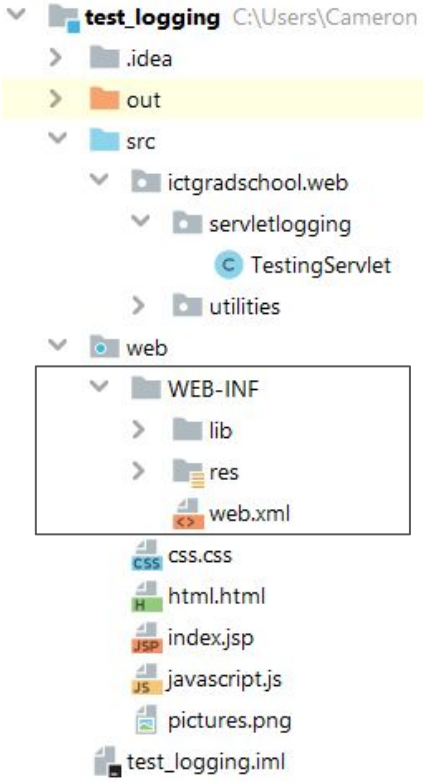


Any files or folders placed in the **web/** folder is made visible, and with the correct URL you are able to access them

- `https://server.tld/test_logging/pictures.png`
- `https://server.tld/test_logging/html.html`
- `https://server.tld/test_logging/some/nested/file`

Your files can be any type and with any name. Folders can have any name, except for two reserved names - **WEB-INF** and **META-INF**

Servlet Directory Structure

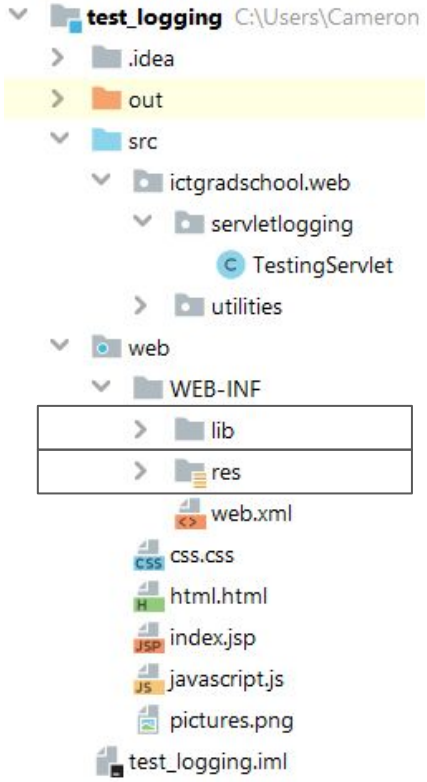


The **`WEB-INF/`** folder is special - anything placed within it will not be included as part of the public document tree of your application

This means that people browsing your site will not be able to see items within this directory

For this reason, it is used to store configurations, libraries, and special files that your servlet application needs, but users of your site do not

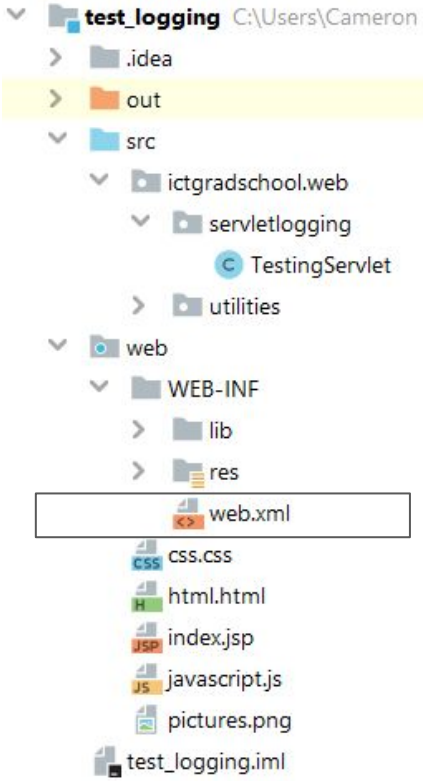
Servlet Directory Structure



`lib/` is the usual folder where libraries are kept. The name is conventional, but could be named anything. To configure the lib folder in IntelliJ:
Right-click | Add as Library....

`res/` is a folder for storing configurations and other resources. Again, the name is a convention, but could be named anything. To tell IntelliJ about these:
Right-click | Mark Directory as | Resources Root

Servlet Directory Structure



`web.xml` is a configuration file that can be used to modify aspects of your servlet application

- Servlet mappings
- Servlet parameters
- Welcome files
- JNDI resources

We will not be using this file often

Starting point

Unlike the Java applications we have been working with up until this point, Servlets take place in an already running Java environment. This means that our programs do not start with a `main()` method, instead our Servlets take the form of classes that extend from `HttpServlet`

HttpServlet uses the Template Method pattern internally through the `service()` method. This method calls hook methods that we can override to implement our server functionality. These hook methods include

```
doGet( HttpServletRequest req, HttpServletResponse resp) {}
doPost(HttpServletRequest req, HttpServletResponse resp) {}
```

Joining the dots ...

`doGet()`, `doPost()`, `HttpServletRequest`, `HttpServletResponse`, these may sound familiar as they all relate to elements we talked about when learning about HTML forms

`doGet()` & `doPost()` are the methods that the Servlet will execute when a `GET` or `POST` request arrives, and each are passed 2 parameters. The `HttpServletRequest` parameter contains all the information about the request including any parameters that were passed to the Servlet, and the `HttpServletResponse` provides functions to allow the Servlet to reply to the request in many different ways

Joining the dots …

On the client-side, forms are bound to Servlets by specifying the method and action attributes of a form

```
<form   method="GET"   action="https://server.tld/MyServlet">
```

On the server-side, the Servlet receives the request and directs it to the appropriate method (`doGet`/`doPost`/…) based on the form type. This method is provided with an `HttpServletRequest` that includes the form data, and an `HttpServletResponse` to control what happens next

The method then determines the result (HTML page/data/error) and replies to the client using the `HttpServletResponse` object methods

Example Servlet

Recall in an earlier lab, you interacted with an ajax service hosted at the URL `https://trex-sandwich.com/ajax/`. This service had a number of endpoints that you used to build functional websites

To interact with these endpoints you created a URL with some GET parameters, `fetch`'ed that URL, then received a JSON document in response.

The JSON document received was produced as the result of running a servlet

Servlet responses

The ajax example responded with JSON, but there are other options

Servlets can respond with:

- Status codes only (no data)
- Unformatted data (plain text)
- Data interchange formats (JSON / XML / YAML / etc)
- Complete web pages
- Redirections to another URL

Servlets can behave differently depending on the verb used (`GET` or `POST`), and parameters sent

Form processing servlets

Form processing servlets often behave differently depending on what HTTP verb is used when communicating with it

GETing from the servlet will cause it to respond with an HTML document containing a form with the same servlet listed in the form `action` attribute

POSTing to the servlet would cause it to instead process submitted form data before redirecting or render a response

This pattern of having a form processing servlet pull double duty both rendering the form and processing submitted forms is a common sight

Example Servlet

```
@WebServlet(name = "Example", urlPatterns = { "/example", "/ex" })
public class ExampleServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                                    throws ServletException, IOException {
        // Indicate what sort of data is being sent back
        resp.setContentType("text/plain");

        // Get an reference to the response stream
        PrintWriter out = resp.getWriter();

        out.println("You sent through the following params:");

        // Loop over all the supplied params and values, writing the values back
        for (String param : req.getParameterMap().keySet()) {
            out.printf("Name: '%s', Value: '%s'", param, req.getParameter(param));
        }
    }
}
```

`@WebServlet` Annotation

What is the `@WebServlet` part of the example?

A Java **Annotation** that provides configuration for the `ExampleServlet`. There are a number of options that can be set, but `name` and `urlPatterns` are the most important

`name` gives the servlet a name that other servlets can use to refer to it

`urlPatterns` specifies one or more URL paths that will link to this servlet. These are relative to the deployed path of the servlet

Deployed paths and urlPatterns

If we were working with a deployed path of:

```
https://trex-sandwich.com/ajax
```

And `StoryServlet` with the `urlPattern` `{ "/story", "/s" }`, then requests to the following URLs would map to `StoryServlet`

```
https://trex-sandwich.com/ajax/story
https://trex-sandwich.com/ajax/s
```

Request Parameters and Forms

When requests originate from forms, data from input elements in forms will be transferred as parameters of the request

We can then access parameters with a range of methods within the servlet

```
<form action="/example01" method="get">
    First name:<br>
    <input type="text" name="firstname">
    <br>
    Last name:<br>
    <input type="text" name="lastname">
    <br><br>
    <input type="submit" value="Submit">
</form>
```

```
public class SimpleServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        String firstName = request.getParameter( s: "firstname");
        String lastName = request.getParameter( s: "lastname");

        request.setAttribute( s: "firstName", firstName);
        request.setAttribute( s: "lastName", lastName);

        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher( s: "/WEB-INF/example01/example01.jsp");
        dispatcher.forward(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

}
```

Request Parameters and Fetch

Also recall from Lecture 09 that parameters can be added to Fetch requests by concatenating them into the URI strings

```
async function chainRequestsAsyncAwait() {
    let articleResponse = await fetch(`https://trex...`);
    let articleJson = await articleResponse.json();
    let userResponse = await fetch(`https://trex...?id=${articleJson.author_id}`);
    let userJson = await userResponse.json();
    console.log(`Title: ${articleJson.title} Author: ${userJson.first_name}`);
}
```

We can access these parameters within the servlet in the same way that we would access parameters that originate from form input fields

Getting request parameters

The `HttpServletRequest` argument received by `doGet`/`doPost`/`etc` methods has functions to get parameter values

`HttpServletRequest.getParameter(String name)`
Get the value of the parameter identified by name

`HttpServletRequest.getParameterNames()`
Get an enumeration of all provided parameter names

`HttpServletRequest.getParameterMap()`
Get a Map of parameter names and values

Responding to requests

Sending any sort of response is as simple as obtaining an output stream and writing to it. Servlets provide access to these via the `HttpServletResponse` argument received by `doGet`/`doPost`/`etc` methods

`HttpServletResponse.getWriter()`

Get a `PrintWriter` stream that we can write responses to

Content can then be written using the `print()`/`println()` methods of the `PrintWriter` stream

Responding to requests

Content written via these streams is just plain text. Applications reading this data may make a best guess as to what the text represents, but we should state what it is supposed to be

`HttpServletResponse.setContentType(String mime)`

Specify the MIME type of the response content

`mime` here could be `"text/plain"`, `"text/json"`, `"text/html"`, or any other valid MIME type

Responding with HTML

Knowing we can use a `PrintWriter` and `setContentType()` to generate responses, we may think to generate HTML using those tools

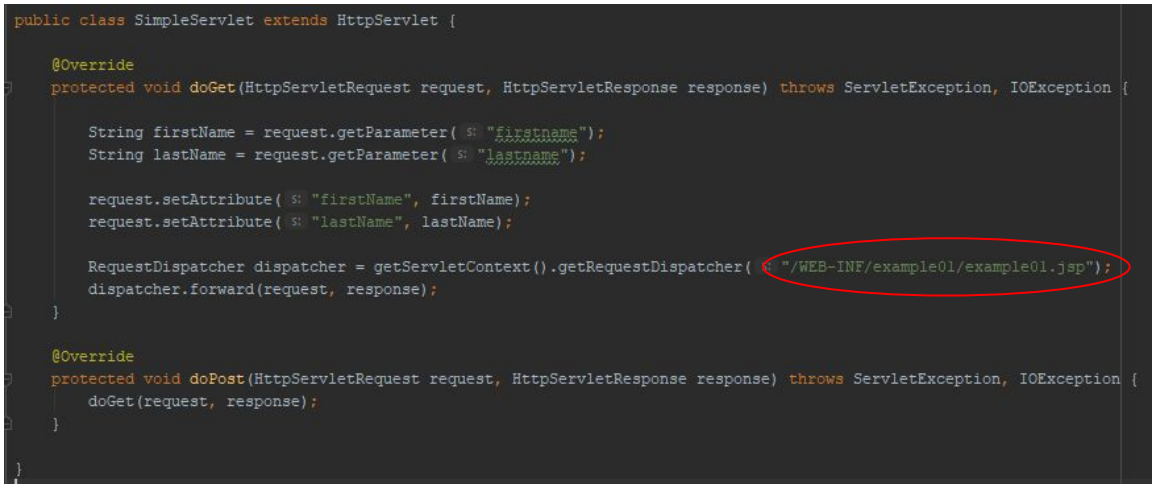
```
resp.setContentType("text/html");
PrintWriter out = resp.getWriter();
out.println("<!doctype html><html><head>");
out.println("<title>" + my_title_var + "</title>");
// ...
```

Resist this urge! It works for simple pages, but complex pages become a nightmare to write and maintain

Responding with HTML

Instead, we want to delegate the HTML rendering to a JSP file.

JSP and the details surrounding it are for a future lecture, but you will use prewritten JSP for exercises today



```
public class SimpleServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        String firstName = request.getParameter( s: "firstname");
        String lastName = request.getParameter( s: "lastname");

        request.setAttribute( s: "firstName", firstName);
        request.setAttribute( s: "lastName", lastName);

        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher( s: "/WEB-INF/example01/example01.jsp");
        dispatcher.forward(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

}
```

JSP File

With server-side web projects JSP files allow us to template HTML server-side and include data from Java

The JSP files in the servlets lab have been written for you so you just need to pass control to them

We will learn more about these later in the course

For today, observe how the examples in the lab project work and connect your servlets with the JSP files in a similar way

```
<%@ page contentType="text/html;c
<%@taglib uri="http://java.sun.co
<%@taglib uri="http://java.sun.co
<html>
<head>
    <title>Title</title>
</head>
<body>
    First name: ${firstName}
    <br>
    Last name: ${lastName}
</body>
</html>
```

Connecting JSP Files and Servlets

To have data from your servlet available in JSP files, you will use the .setAttribute() method of the request object.

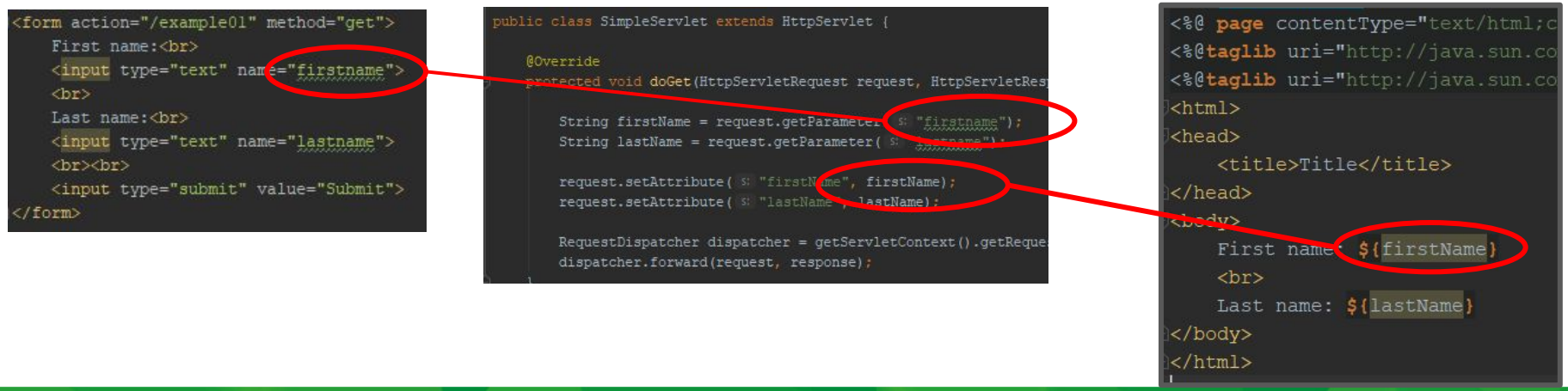
We will learn more about JSP and request attributes in later lectures but for today's lab make sure to observe how the example servlets and JSP works.

```java
public class SimpleServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletRes

        String firstName = request.getParameter( s: "firstname");
        String lastName = request.getParameter( s: "lastname");

        request.setAttribute( s: "firstName", firstName);
        request.setAttribute( s: "lastName", lastName);

        RequestDispatcher dispatcher = getServletContext().getReque
        dispatcher.forward(request, response);
    }
```

```jsp
<%@ page contentType="text/html;c
<%@taglib uri="http://java.sun.co
<%@taglib uri="http://java.sun.co
<html>
<head>
    <title>Title</title>
</head>
<body>
    First name: ${firstName}
    <br>
    Last name: ${lastName}
</body>
</html>
```

Forms -> Servlet -> JSP/HTML

There will often be a lot more happening in between but make sure to think about how data is connected across forms, servlets and JSP files



Learning the Relevant APIs

In practical terms, developing Servlets comes down to learning the APIs to a set of classes. The core classes that you will see in all Servlets are

`HttpServlet`
`HttpServletRequest`
`HttpServletResponse`

More advanced classes that we will discuss in the future include

`HttpSession`
`Cookie`