Objectives

- Review: HTML Forms
- Intro to Java Server-side Web Technology

1. Start eclipse
2. Start a new workspace

Review: Bootstrap

- What is Bootstrap?
- What are the pros and cons of using Bootstrap?

Review: HTML Forms

1. Start eclipse
2. Start a new workspace

- What attribute is required in a `form` tag?
  - What attribute is optional?
- What ***attribute*** do we use to create different types of `input`?
- How do we distinguish between input data?
- How do we “group” radio buttons and checkbox buttons?
- What tag do we use to improve usability of our radio buttons and checkboxes?
- What are the differences between “get” and “post” requests?
  - When should we use “get” vs “post”?

Review: Java

- What is Java?
- How do we write Java code?
  - What is the syntax of Java?
  - What are coding conventions of Java?
- What are differences between Java and Python?
- What is the class path?
- How are classes organized in Java?
- How do you compare Strings in Java?
- What does it mean if one class *extends* another class?
- Compare and contrast *classes* vs *interfaces* vs *abstract classes*
- How do you find out what you can do in Java, e.g., what classes are available?
- What is Eclipse?

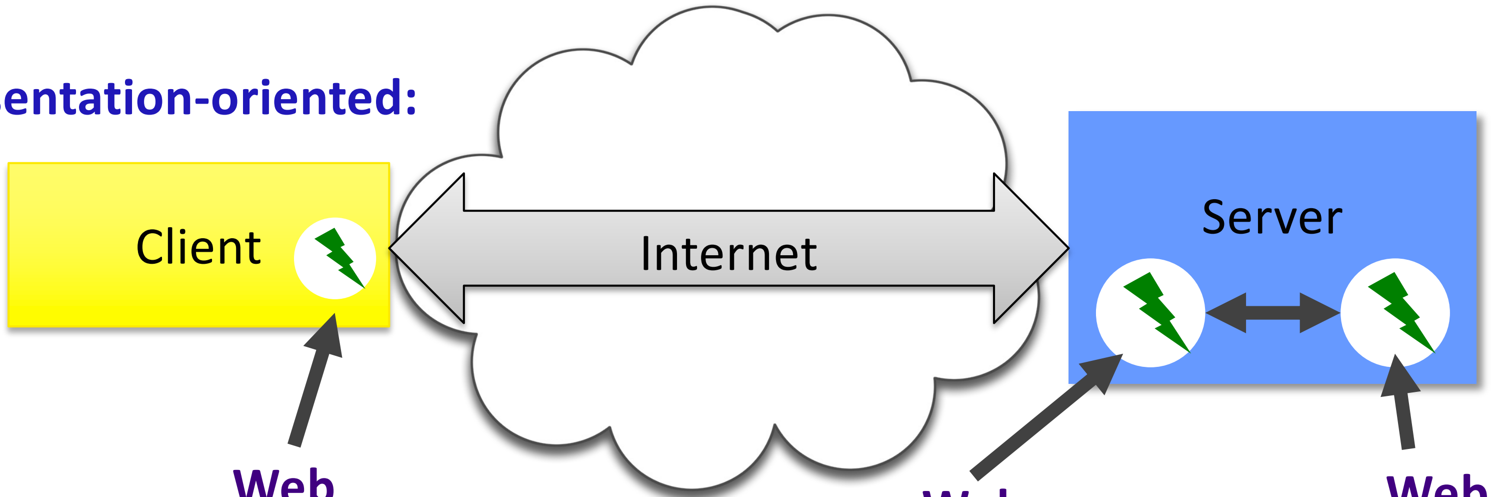
Java

- Object-oriented language
  - All code is defined within classes
  - Made up of objects; call methods on the objects
- Syntax highlights
  - object.methodname();
  - Curly braces around blocks of code
  - Conditions are in parentheses
- Classpath: where to look for Java classes
- Organized in *packages*
- Java API/Javadocs
- Eclipse is an IDE with lots of tools for Java

INTRODUCTION TO SERVER-SIDE PROGRAMMING

Architecture of the Web

Presentation-oriented:



Web Browser

- Makes requests, renders responses
- Executes JavaScript, client-side code

Web Server

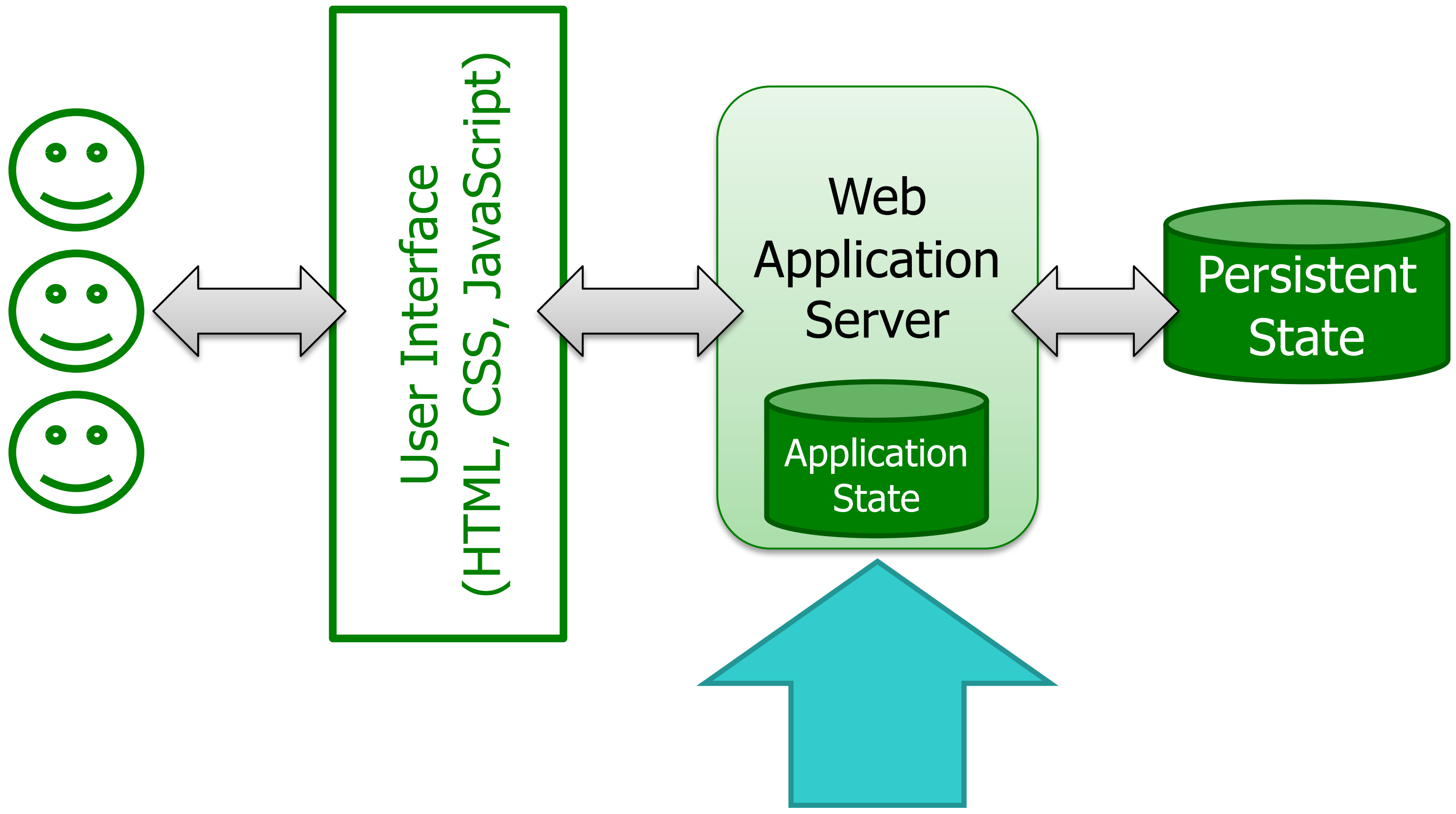
handles static requests

Web Application Server

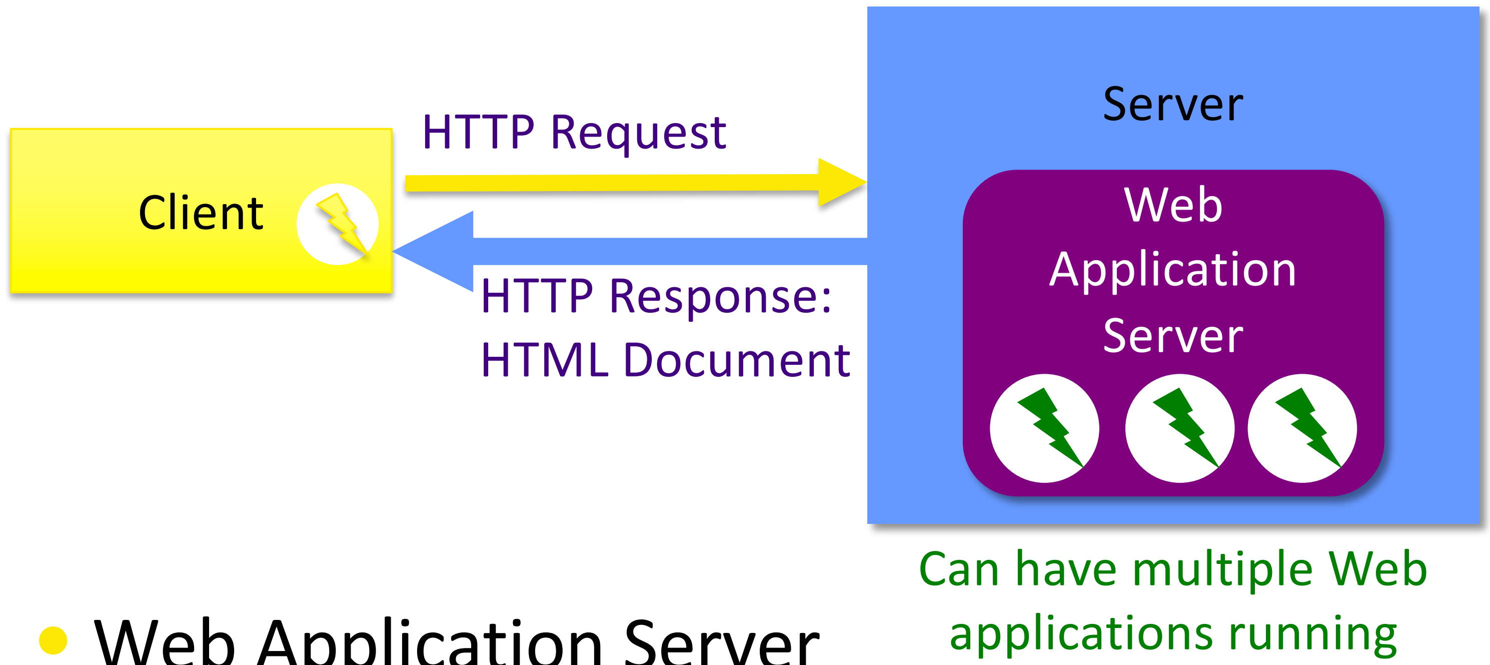
handles ***dynamic*** requests

Service-oriented Alternative:
Client is another server/non-user computer

Web Application Architecture



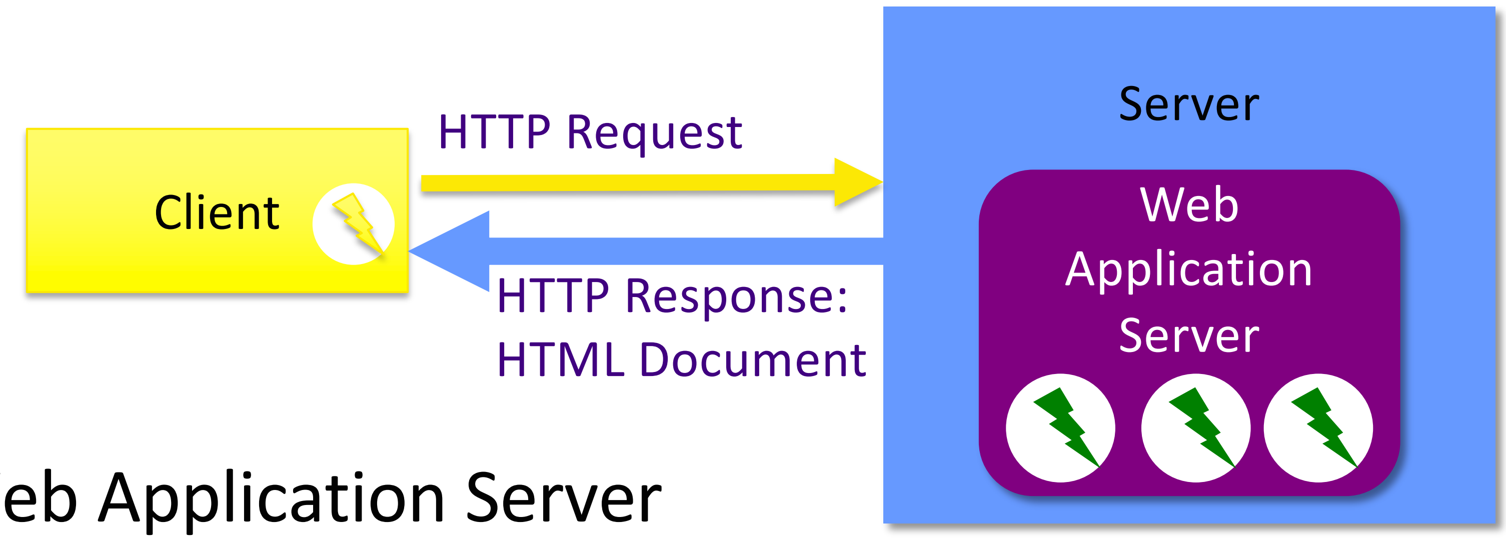
Java-based Web Application Server



Client
HTTP Request
HTTP Response: HTML Document
Server
Web Application Server
Can have multiple Web applications running

- Web Application Server
  - ***Container*** to run the Java-based web applications
  - Typically listens on port 8080 (rather than 80)

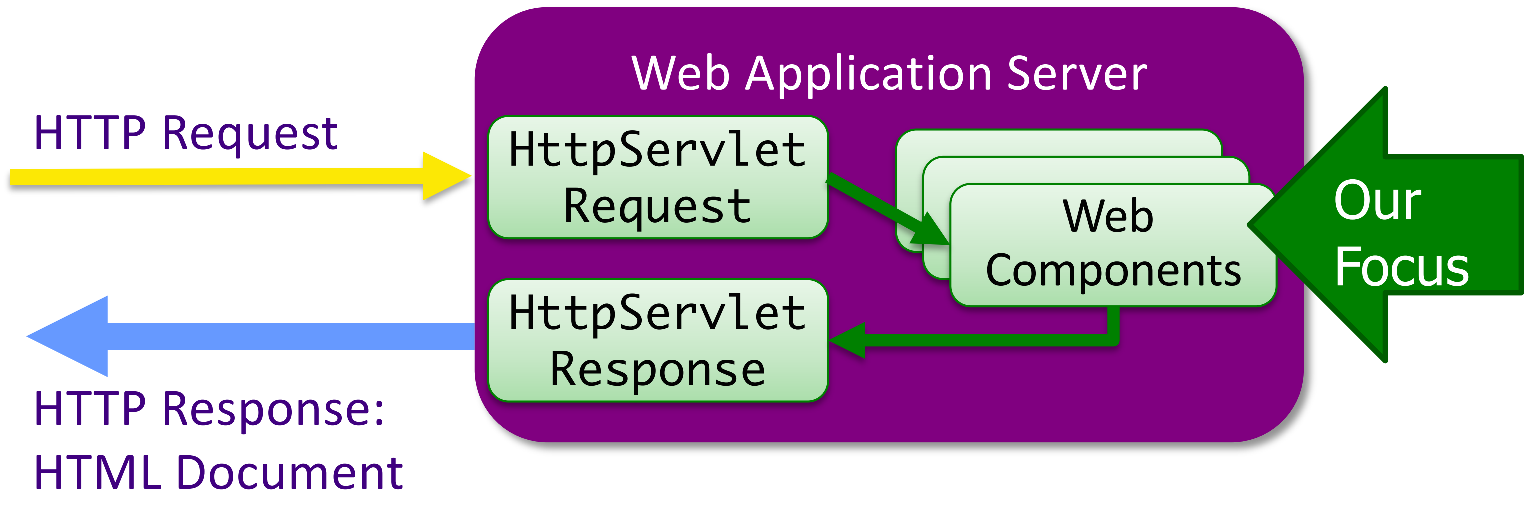
Java-based Web Application Server

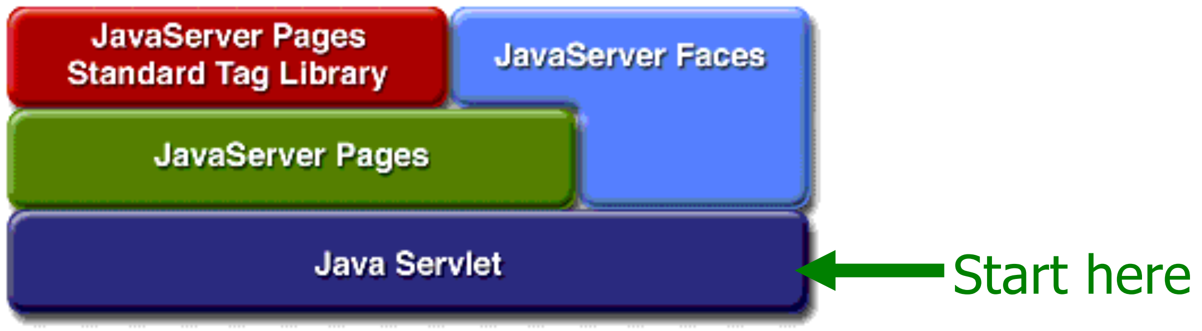


Can have multiple Web applications running

- Web Application Server
  - ➢ Parses request, including data
  - ➢ Executes request
  - ➢ Returns response (often an HTML document)
    - May do other things, like send email, …

Request Handling in Java





Servlets

- A Java class that extends the functionality of web servers
  - Processes requests on server
  - Sends results (typically as an HTML file) back to client
- In `jakarta.servlet.*` packages
  - Part of Java Enterprise Edition (EE), as a separate download
  - Eclipse for EE development (Web Tools Platform)
- Java's answer to CGI (Common Gateway Interface)
- Portable, more secure (no buffer overflows)
- Supported by many major Web servers
  - E.g., Apache Tomcat, Jetty, WebSphere, etc.

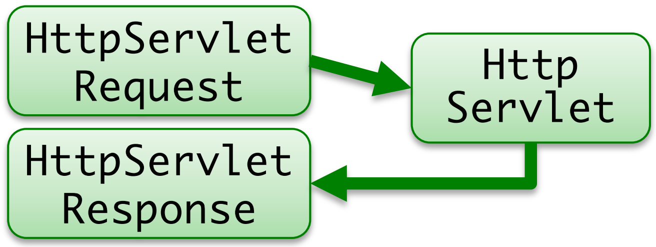
The `Servlet` Interface

Review from CSCI209: What is an *interface*?

- `jakarta.servlet.Servlet`
- All servlets implement the `Servlet` interface
  - `HttpServlet`, `GenericServlet`, `FacesServlet`
  - Web application server invokes many methods of `Servlet` automatically

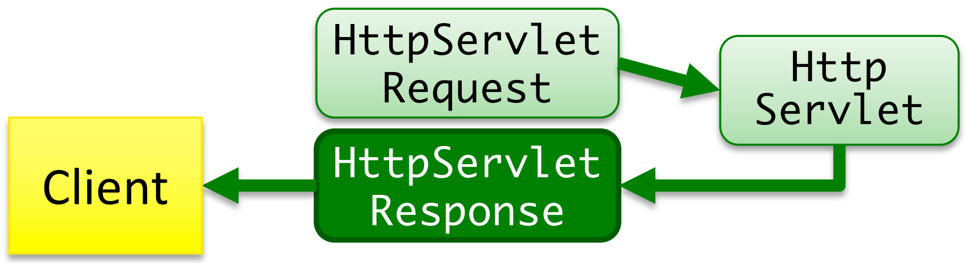
The `HttpServlet` Class

- Web-based servlets typically *extend* `HttpServlet`
  - Implements `Servlet` interface
- `HttpServlet` implements the `service` method
  - Parameters
    - `HttpServletRequest` - from the client
    - `HttpServletResponse` - to the client
  - `service` calls the respective method (e.g., `doGet` or `doPost`) in response to a HTTP GET or POST request
- Recall:
  - **GET** - data encoded in URL
    - Request a resource (file) or retrieve data
  - **POST** - data encoded in body of message
    - Upload data; processing; hide data from URL



`HttpServletResponse`

- Provides output streams and methods to write data to the client



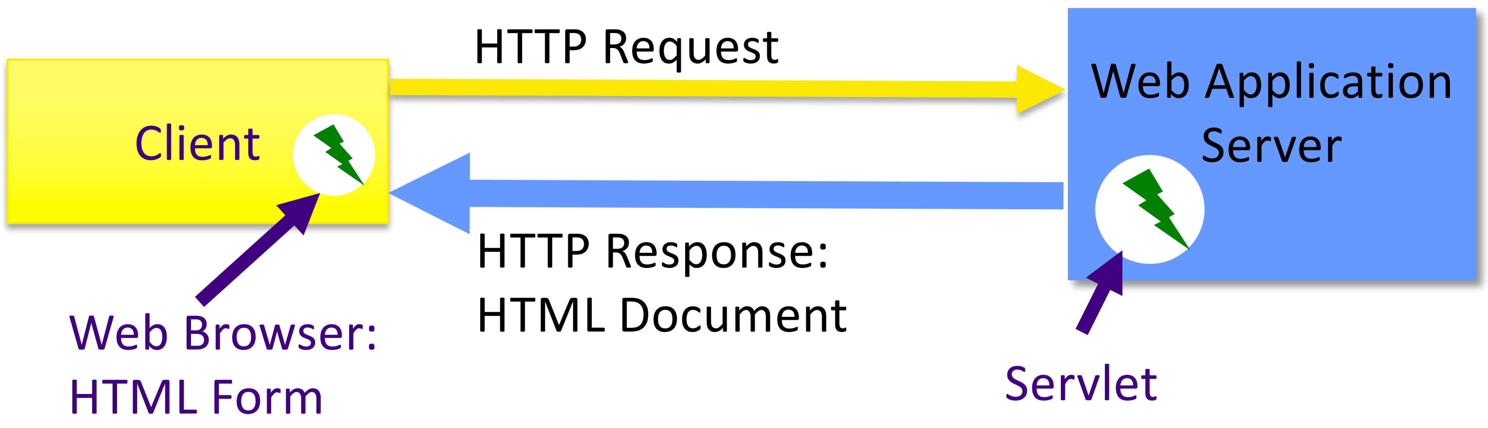
- Methods:
  - `ServletOutputStream getOutputStream()`
  - `PrintWriter getWriter()`
  - `void setContentType(String)`

CSCI209 Flashback: Difference between *streams* and *writers* in Java?

`HttpServletResponse` methods

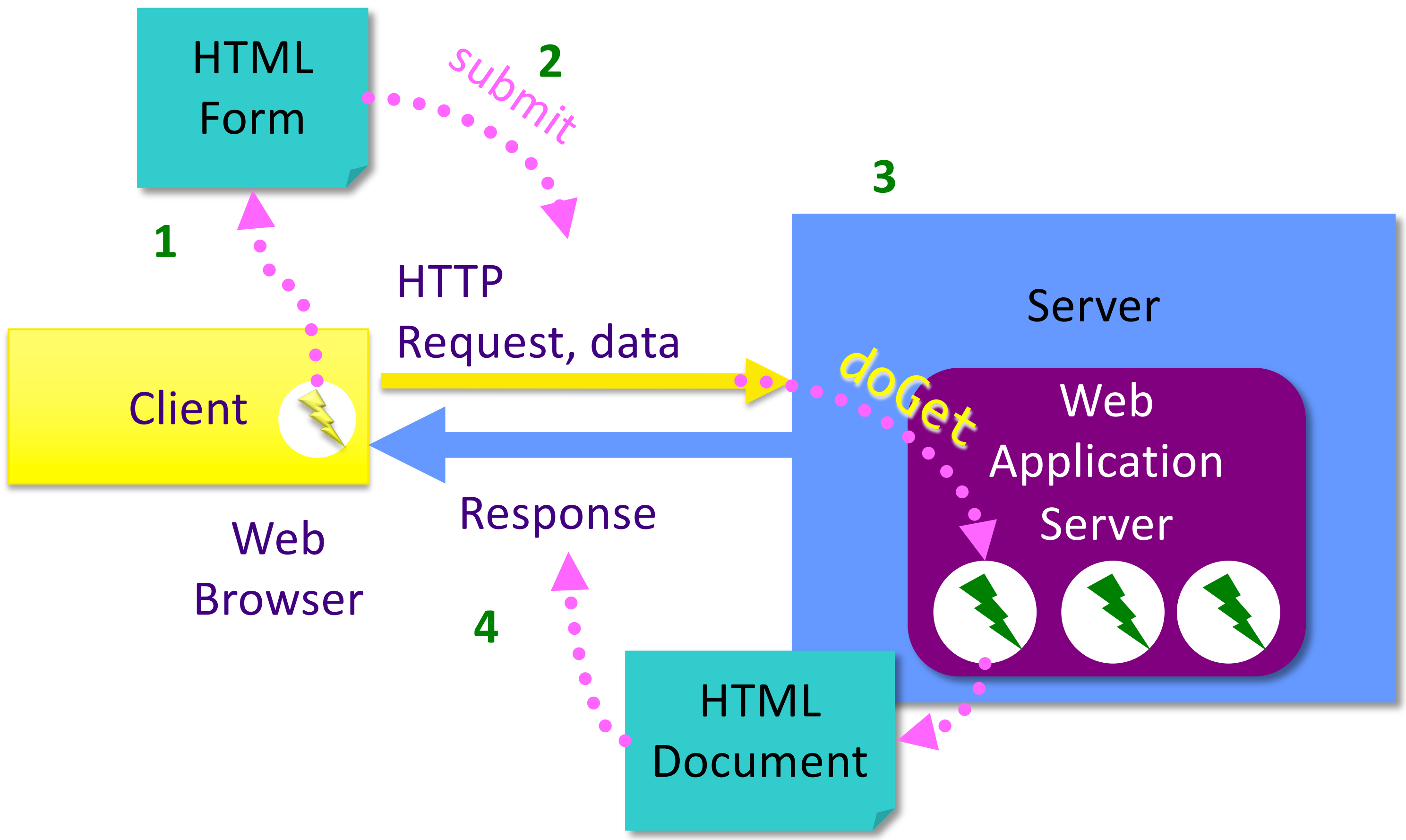
- `ServletOutputStream getOutputStream()`
  - Obtains a byte output stream that enables the servlet to send *binary data* to the client
- `PrintWriter getWriter()`
  - Obtains a text writer that enables the servlet to send *character data* (text) to the client
- `void setContentType(String)`
  - Specifies the MIME type of the response
    - Browser knows what it received and how to format it
  - `text/html` specifies an HTML document

Example Communication with Servlet



- HTML page with a form containing a submit button
  - Triggers client's request
- When the button is pressed, browser sends the servlet a GET request

Example Servlet Flow



To Generate a Response

```
void doGet(HttpServletRequest request, HttpServletResponse response)
```

- `doGet` method needs to
  - Obtain an output *writer* to write back to the client
  - Generate/write the HTML page to the client using the writer
  - Close the writer

How do we implement these steps?

Accepting Certain Types of Requests

- We can design the servlet to **only** accept/handle GET requests
- Example:
  - Override the `doGet` method
  - Return an error inside of `doPost`
- Or could do the opposite: only accept/handle POST requests

HTTP Response Errors

- HttpServletResponse has a method for returning errors and fields that define errors codes

```
void sendError(int statusCode [, String msg])
```

- Example status code fields:
  - SC_HTTP_VERSION_NOT_SUPPORTED
  - SC_METHOD_NOT_ALLOWED
  - SC_NOT_IMPLEMENTED

- Tomcat
- Eclipse/Tomcat
- Using Eclipse
  - HTML, CSS
  - Start servlets

IN-CLASS WORK

`web.xml` File

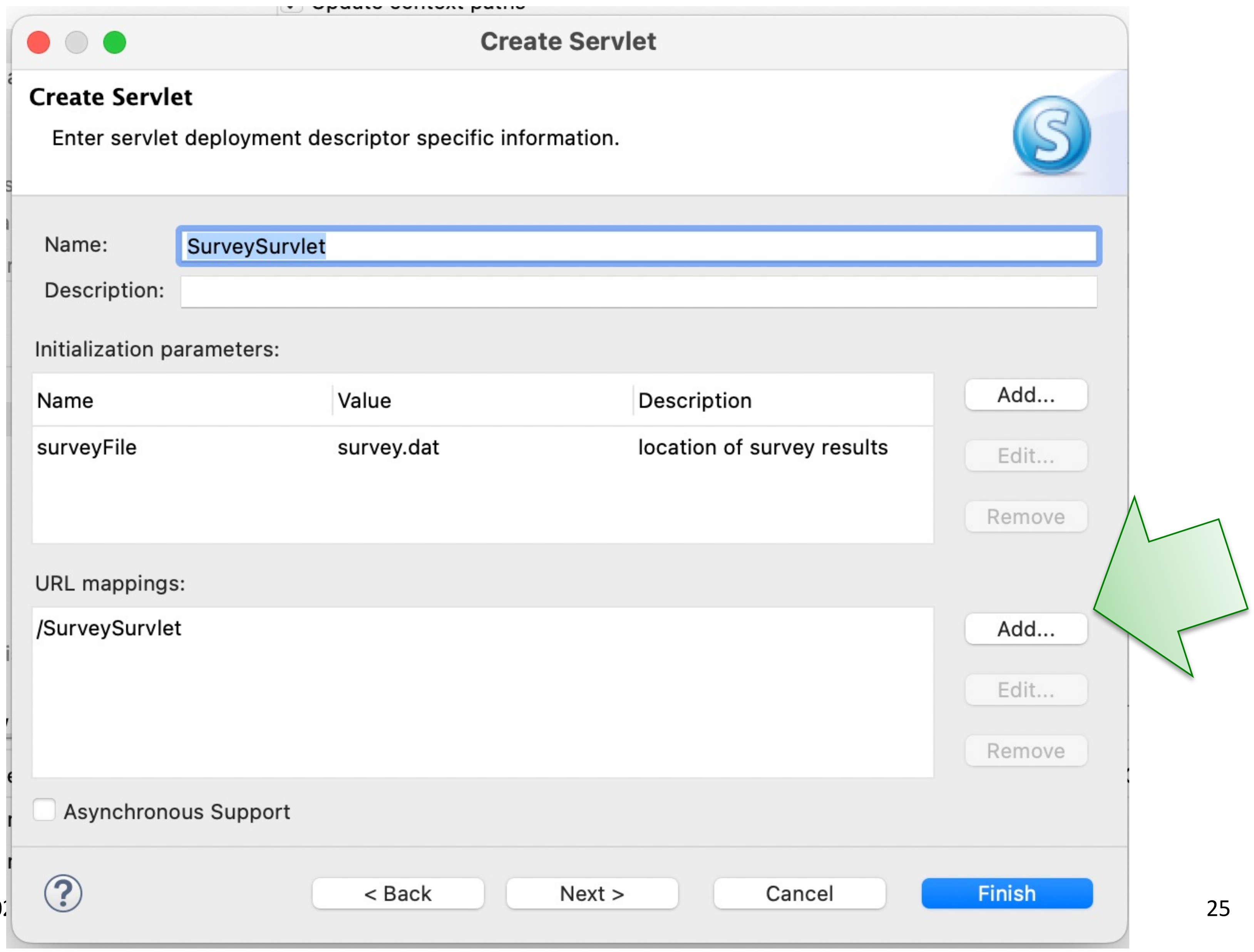
- Describes how to deploy the web application
- XML file
  - Used for data
  - Marked up with *elements*
  - Rule: must close most recently opened tag, attributes in quotes

```
<tag attr="value">
       Content
</tag>
```

- DTD: Document Type Definition
  - Define elements that can be in a particular XML document
  - Includes specification of attributes, nesting

Eclipse + web.xml

- When you create a new servlet, Eclipse automatically updates web.xml with
  - The servlet name
  - The URL mapping to the servlet (the URL /ServletName → package.ServletName.java
- (You may have seen this set up if you clicked “next” when you create the servlet)



Create Servlet

Enter servlet deployment descriptor specific information.

Name: SurveySurvlet

Description:

Initialization parameters:

Name	Value	Description
surveyFile	survey.dat	location of survey results

URL mappings:

/SurveySurvlet

Asynchronous Support

Eclipse + web.xml

- When you create a new servlet, Eclipse automatically updates web.xml with
  - The servlet name
  - The URL mapping to the servlet (the URL /ServletName → package.ServletName.java
- (You may have seen this set up if you clicked “next” when you create the servlet)

Common Issue: Eclipse does *not* update web.xml as you make changes to class names, packages, etc., so ***YOU*** need to make changes to web.xml yourself.

Annotations

- As of Servlets 3.x, we can easily configure a web application using ***annotations***
- Old way: all configuration in web.xml
- Now: Annotations provide defaults, can be overridden in web.xml
- Example:

```
@WebServlet("/superdeeduper")
public class MyServlet extends HttpServlet {
```

  - Means the URL pattern "/superdeeduper" maps to this servlet (servlets.MyServlet)

Handling Data

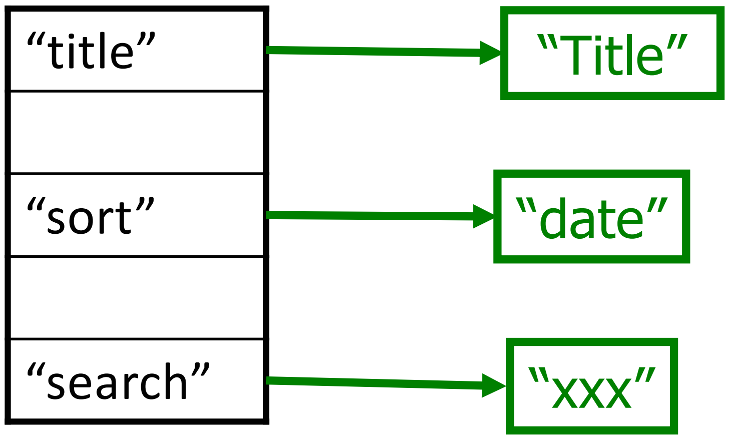
Relate back to the form/ Input names

- So far, we haven't done anything with data that comes with the request

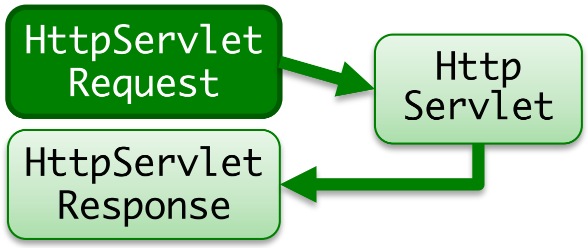
Requests for a digital publication library:

```
GET /search?search=xxx&sort=date&title=Title
```

- Data is stored in a hashtable-like object



HttpServletRequest



- Provides input streams and methods to read data from the client
- Methods:
  - `String getParameter(String paramname)`
    - Returns the value of a request parameter
      - `null` if parameter doesn't exist
  - `String[] getParameterValues(String paramname)`
    - Returns an array of Strings containing the values for a specific request parameter
  - `Enumeration<String> getParameterNames()`
    - Returns the names of all parameters passed in the request

`HttpServletRequest` Methods

Requests for a digital publication library:

```
GET /simple-search?search=xxx&sort=date&title=Title
```

- `request.getParameter("title")`
  - Returns "Title"
- `request.getParameterValues("sort")`
  - Returns ["date"]
- `request.getParameterNames()`
  - `Returns Enumeration { "search", "sort", "title" }`

Return strings

An Example: A Pet Survey

- An HTML form that asks the user for their favorite type of pet
- After user submits the form, the server sends back the current results of the survey
- Uses object serialization to write to/read from file

Deployment: WAR files

- Web Archives
  - Analog to JAR files
  - Bundles together all the code, files for the web application
- Copy into webapps directory of web application server
  - Server will automatically extract files and run
- Can export WAR files from Eclipse
  - For your submission, export the WAR
    - Make sure you check the box for “Export source files”

Breaking Problems Down

- When you started programming, you would write a few lines, run the program, and see if it was right. Repeat.
- With web app programming, you can't do that
  - A little harder to figure out the chunks you can do before testing
  - BUT, you should find those chunks. Don't try to complete everything all at once.

TODO

- Complete Lab 4
  - Due tonight at midnight
- Your own web page
  - Due Monday at midnight
- Read “Quality Attributes of Web Software Applications”
  - Writeup on Canvas
  - Due Tuesday @ midnight