Servlets and JSP

A Java Servlet is a software component that extends the capabilities of a server. JSPs are a collection of technologies that are used to create dynamically generated web pages based on HTML, XML, SOAP, etc.

Outline

1. Installing Tomcat
2. Servlets
   - Introduction
   - Lifecycle
   - Coding With Servlets
3. JSP
   - Introduction
   - Lifecycle
   - Creating JSPs
4. Maven Project

Prerequisites

1. JDK Version 11
   - If on Windows, make sure **java is on your path**: How to set up Path
2. Eclipse Download **(Windows Users)**
3. STS Download **(Mac/Linux Users)**
   - Click on download for STS on Eclipse
4. Apache Tomcat
   - Download Tomcat 9, latest version
   - Pick the Core zip file and unzip it inside your c drive or documents folder

Introduction to Servlets

Why do we have Servlets?

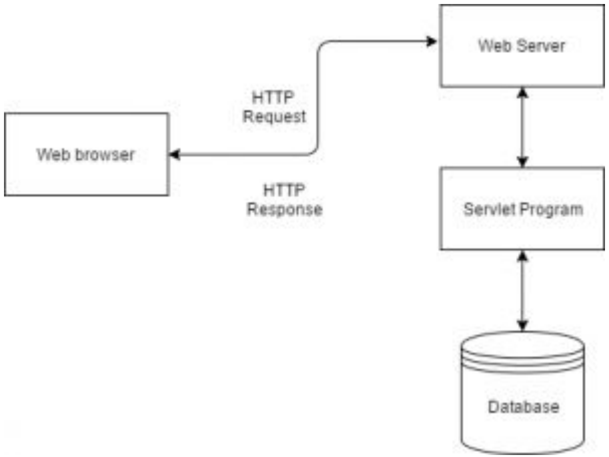
- ➔ We want to have ***dynamic web pages***
  - ◆ Change contents on site based on client requests
  - ◆ Change contents based on time/events
- ➔ **Servlets** can handle/process requests from a web server and generate a response that can dynamically change content on our web pages
- ➔ Before, ***CGI (Common Gateway Interface)*** was used, but was more expensive, slower, and with weaker security

What are Servlets?

- ➔ **Servlets** in basic terms, receive requests, process them, and reply back with a response
- ➔ Receive requests from web server (front end)
- ➔ Then return back a response
  - ◆ Change to be made on front end
  - ◆ Acknowledge request has been fulfilled
- ➔ **Servlet Container (Servlet Engine)** loads the servlet and provides the runtime environment
  - ◆ Contains other services to manage servlet lifecycle, provide security, manage sessions, etc.

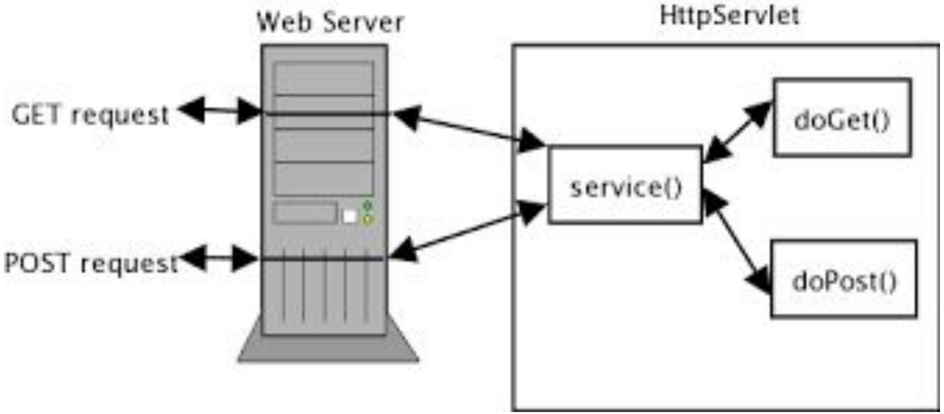
Steps to Executing a Servlet

1. Clients send request to web server
2. Web server receives request
3. Web server passes request to corresponding servlet
4. Servlet processes request and generates a response as an output
5. Servlet sends response back to the web server
6. Web server sends response back to client and client browser displays it on the screen



Servlet Classes

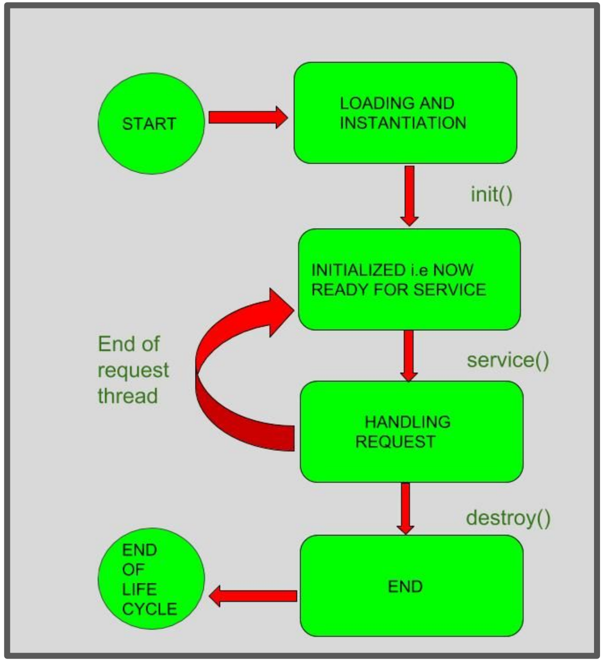
- ➔ All Servlet classes implement ***javax.servlet.Servlet*** interface
- ➔ Most common used implementation is **HttpServlet**



Servlet Lifecycle

Servlet Lifecycle

- Lifecycle managed by Servlet Container
- Steps:
  - ***Load Servlet Class***
  - ***Create Instance of Servlet***
  - ***Call the servlets init() method***
  - ***Call the servlets service() method***
  - ***Call the servlets destroy() method***



Servlet Lifecycle

Load Servlet Class

- ➔ Servlet container loads Servlet class definition (just like any other class)

Create Instance of Servlet

- ➔ When the servlet class is loaded, the servlet container creates an instance of the servlet. Typically, only a single instance of the servlet is created, and concurrent requests to the servlet are executed on the same servlet instance. This is really up to the servlet container to decide, though. But typically, there is just one instance.

Call the Servlets init() Method

- ➔ When a servlet instance is created, its init() method is invoked. The init() method allows a servlet to initialize itself before the first request is processed. You can specify init parameters to the servlet in the web.xml file

Servlet Lifecycle

Call the Servlets service() Method

- ➔ For every request received to the servlet, the servlets service() method is called. For HttpServlet subclasses, one of the doGet(), doPost() etc. methods are typically called. As long as the servlet is active in the servlet container, the service() method can be called. Thus, this step in the life cycle can be executed multiple times.

Call the Servlets destroy() Method

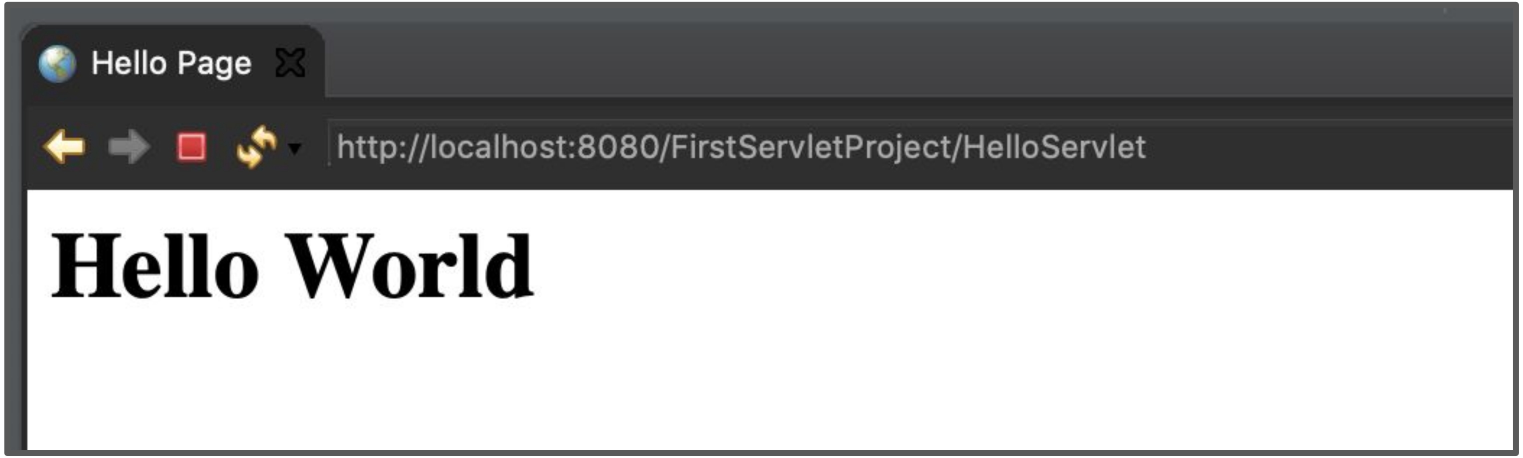
- ➔ When a servlet is unloaded by the servlet container, its destroy() method is called. This step is only executed once, since a servlet is only unloaded once. A servlet is unloaded by the container if the container shuts down, or if the container reloads the whole web application at runtime.

HTTP Requests

- HttpServlet class reads **HTTP requests** like:
  - ***GET*** = retrieve data
  - ***POST*** = create data
  - ***PUT*** = update data
  - ***DELETE*** = delete data
  - And more…
- To send a request e.g. HTTP GET, extend HttpServlet class and override doGet() method

Coding With Servlets

Creating Our First Servlet Project



http://localhost:8080/FirstServletProject/HelloServlet

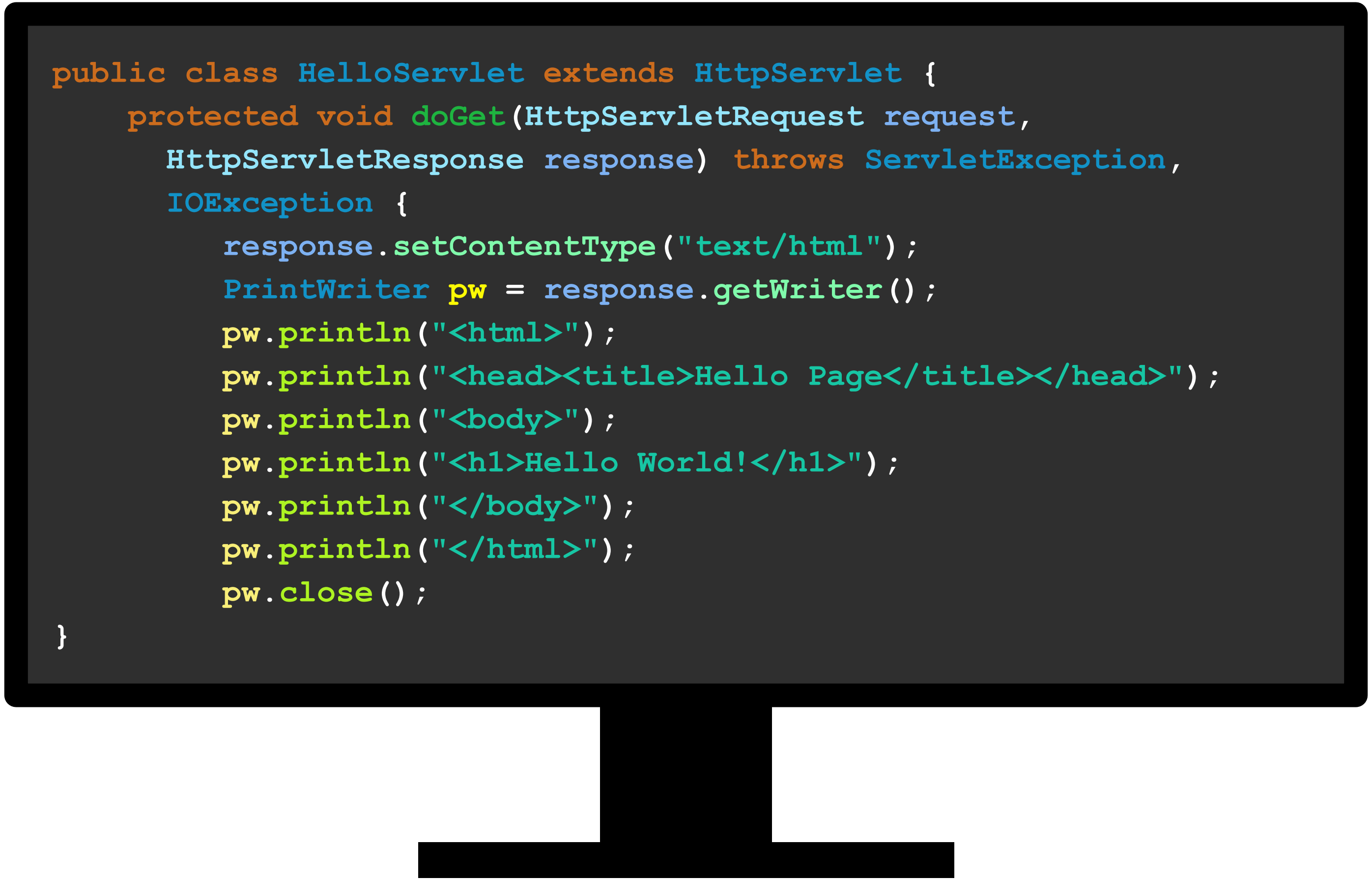
Hello World

Setting up Tomcat Server

- ➔ In Eclipse, go to **Window** → **Show View** → **Servers**
  - ◆ If Servers not shown, click on **Other** and search for Servers in window that pops up
- ➔ Now go to **File** → **New** → **Other** → Search for **Server** → **Next** → select **Tomcat v9.0** → **Next**
- ➔ Click on **Browse** to ***select folder where Tomcat is,*** click **Finish**
- ➔ You should now see the Tomcat server on the Servers tab
- ➔ Double click on the server, a page should open
- ➔ Under **Server Locations**, select **Use Tomcat Installation** and save this change (do a Ctrl+S)
- ➔ Right click on Tomcat and Start the server
- ➔ Open your browser, type into search bar, http://localhost:8080/, you should see a page that says Apache Tomcat

Setting Project Using Servlets

- ➔ Create a new ***Dynamic Web Project, give it a name***, click **Next** -> **Next**
  - ◆ Check the box to ***Generate web.xml*** then click **Finish**
- ➔ Add the **servlet-api.jar** into your build path (located in lib folder for tomcat)
- ➔ ***Create new package*** under **src** with **new class** in this package that ***extends HttpServlet*** and set it up (can also look for and create **new Servlet**)
  - ◆ ***Update the web.xml*** with the servlet tags for new Servlet ***or add @WebServlet() annotation***
- ➔ Right click on Servlet file -> **Run As** -> **Run on Server** -> select **Tomcat v9.0** -> **Next** -> select your project -> **Finish**

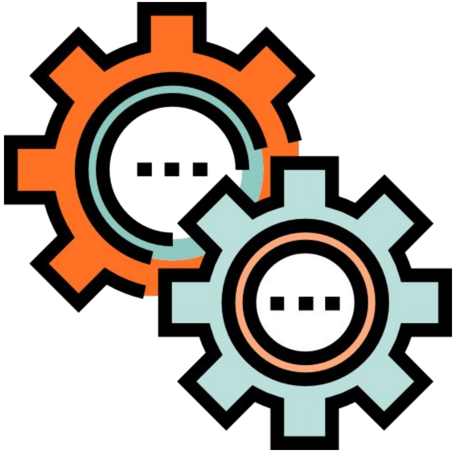


```
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException,
      IOException {
         response.setContentType("text/html");
         PrintWriter pw = response.getWriter();
         pw.println("<html>");
         pw.println("<head><title>Hello Page</title></head>");
         pw.println("<body>");
         pw.println("<h1>Hello World!</h1>");
         pw.println("</body>");
         pw.println("</html>");
         pw.close();
}
```

Init, Service, and Destroy Method

- When we load our servlet to be used, **init() method** is called before it processes any request
- The **service() method** processes requests as many times as needed
  - Will call methods like **doGet()** or **doPost()** to process those requests
- When servlet unloaded, **destroy() method** is called
  - Can be used to wrap up any processes or close connections

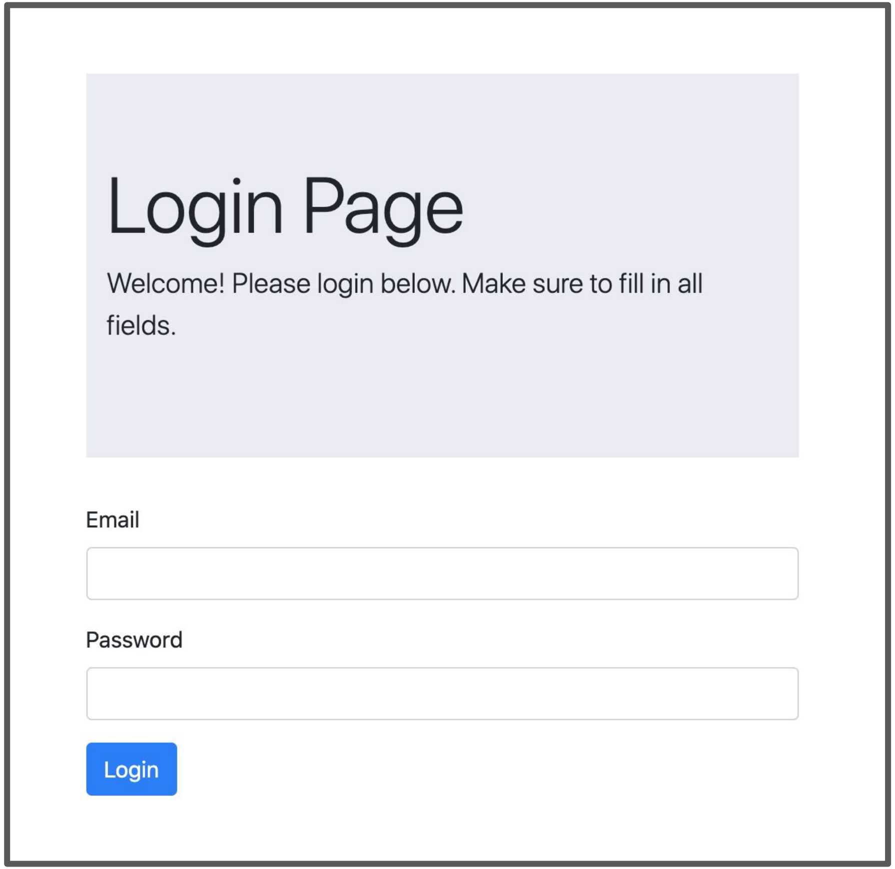
web.xml



- Within the ***WEB-INF*** folder
- The web.xml file ***enables servlet to function***
- Contains instructions for tomcat to recognize the servlet
- **<servlet>**
  - **<servlet-name>** → names and identifies servlet
  - **<servlet-class>** → name of our servlet class and package it’s within
- **<servlet-mapping>**
  - Path to where servlet will be loaded and display dynamic web page
  - Contains **<servlet-name>** and **<url-pattern>** for path to servlet

Simple Login Project

- ➔ Set up a login page
- ➔ Successful login:
  - ◆ Email ends with **@cognixia.com**
  - ◆ Password is **“123”**
- ➔ Create a form that sends request to servlet
- ➔ Servlet response whether login is successful or not



Login Page

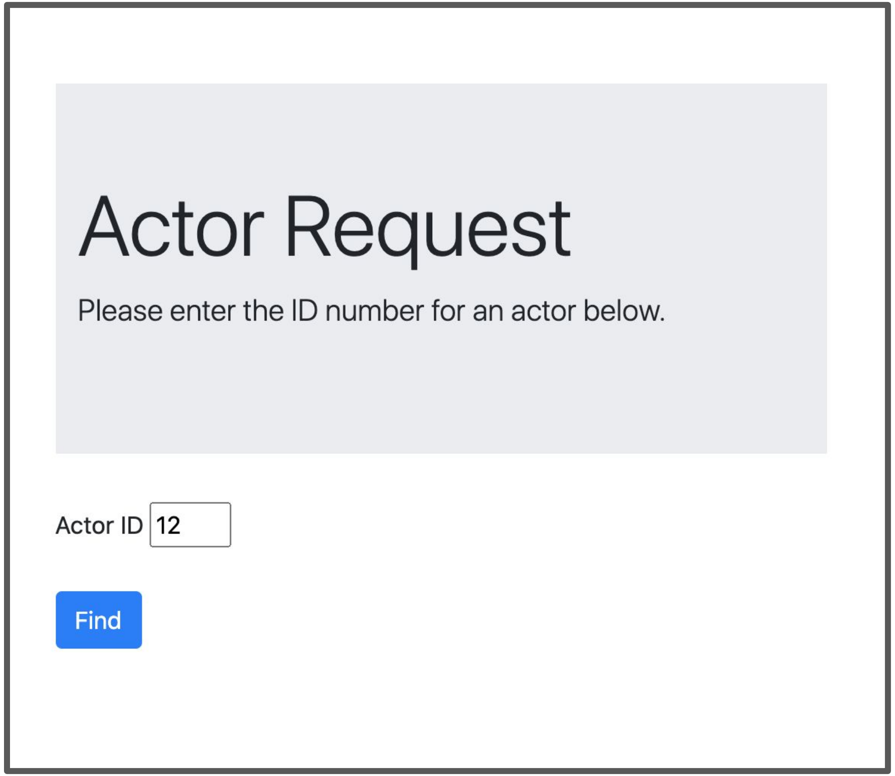
Welcome! Please login below. Make sure to fill in all fields.

Incorporating JDBC to Servlet Projects

- Create Dynamic Web Project
- Right click on project, **Build Path** → **Configure Build Path** → **Library** → click on ***Classpath*** → **Add External JARs** → add in ***MySQL Connector JAR***
- On left panel, click on **Deployment Assembly**
  - If window pops up to save modifications, click Apply
- Click **Add** → **Java Build Path Entries** → **Next** → click on the ***MySQL Connection JAR*** → **Finish**
- Click **Apply and Close**

Find Actor Project

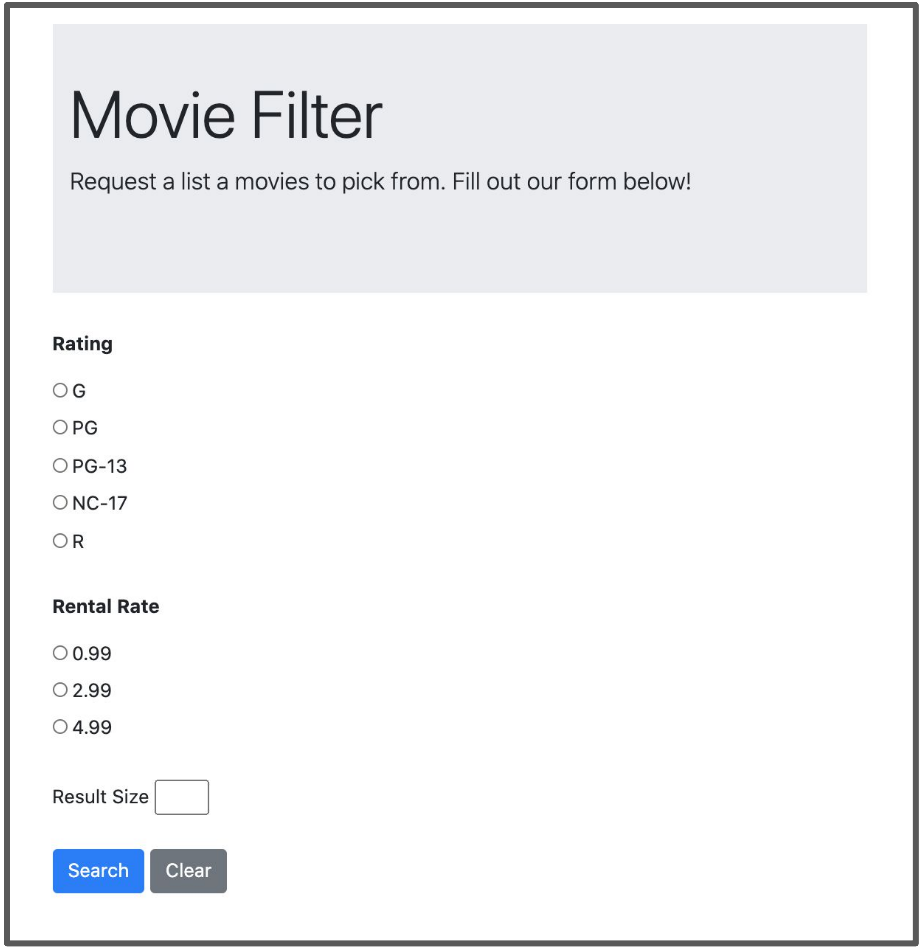
- ➔ Set up a page to get the name of an actor based on their ID
  - ◆ Note: biggest ID from actor table within sakila is 200
- ➔ Connect our project to sakila database
- ➔ Make sure input can only be a number between 1 to 200



Actor Request

Please enter the ID number for an actor below.

Actor ID 12



Movie Filter

Request a list a movies to pick from. Fill out our form below!

Rating

○ G

○ PG

○ PG-13

○ NC-17

○ R

Rental Rate

○ 0.99

○ 2.99

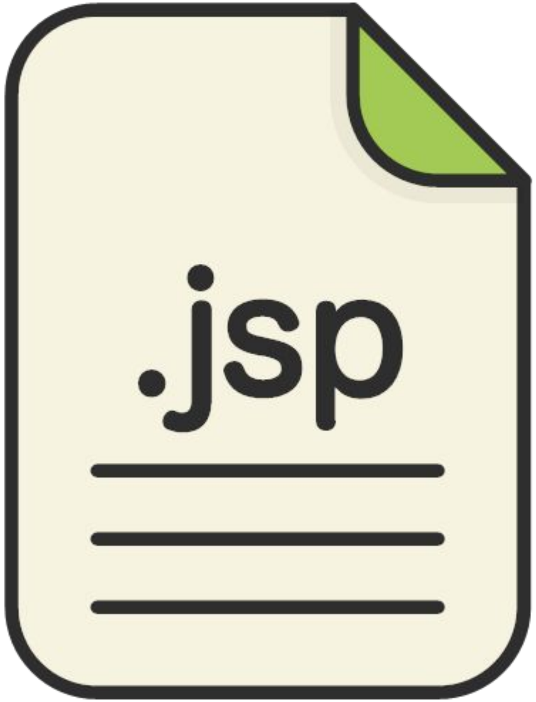
○ 4.99

Result Size □

Exercise: Find The Film

- → Set up a form like on the left that finds all films who have the ***same rating and rental rate***
- → These are both radio buttons
- → Add a ***result size input that will only return first 1-20 films*** with these specifications
- → You only need to ***return the title and description of the films*** selected

Introduction to JSPs

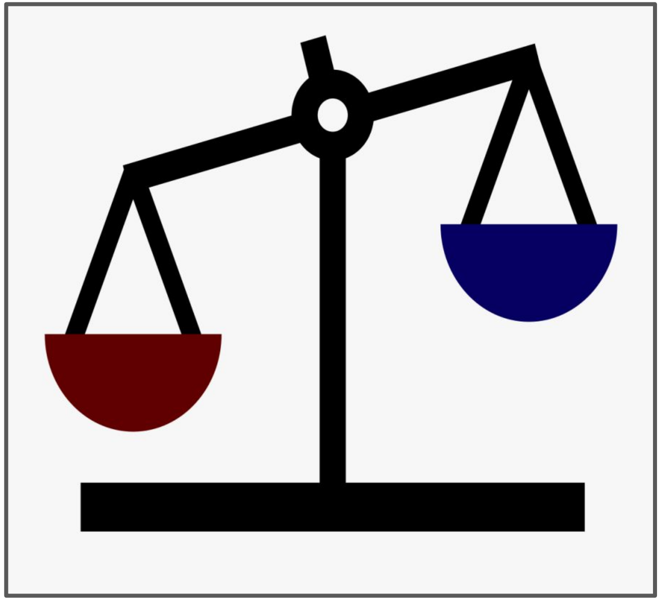


Java Server Pages (JSP)

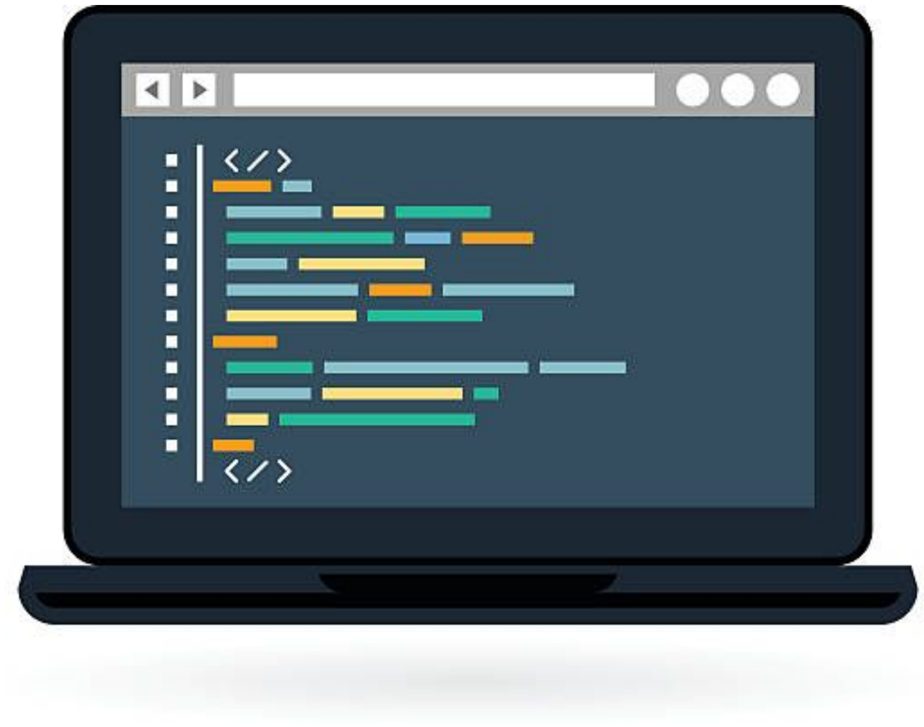
- Java Server Pages (JSP) is a technology to develop web pages
  - Can inserting Java code into HTML pages
  - Tags where java code can be included: `<% ----java code----%>`
- Can consist of HTML or XML (or combination) with JSP actions and commands
- *Used to create dynamic content like forms and registration pages*
  - Fields like dropdowns and checkboxes can contain values fetched from a database
- Used to access JavaBeans objects
- Share information across pages using request and response objects
- Separates view layer with the business logic of web application

Main Advantages of JSPs

1. Capable of handling exceptions
2. Easy to use and learn
3. Has tags which are easy to use and understand
4. Has implicit objects, reduces code

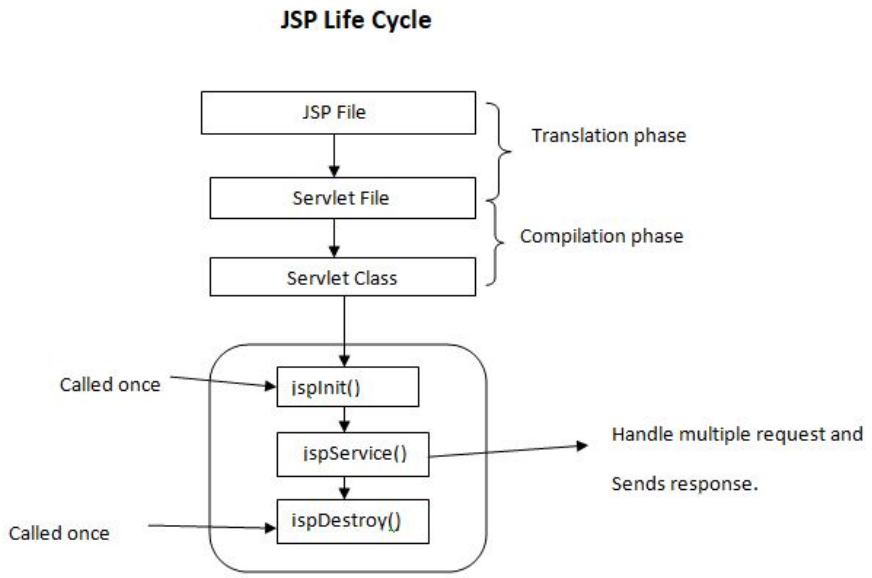


JSP Lifecycle



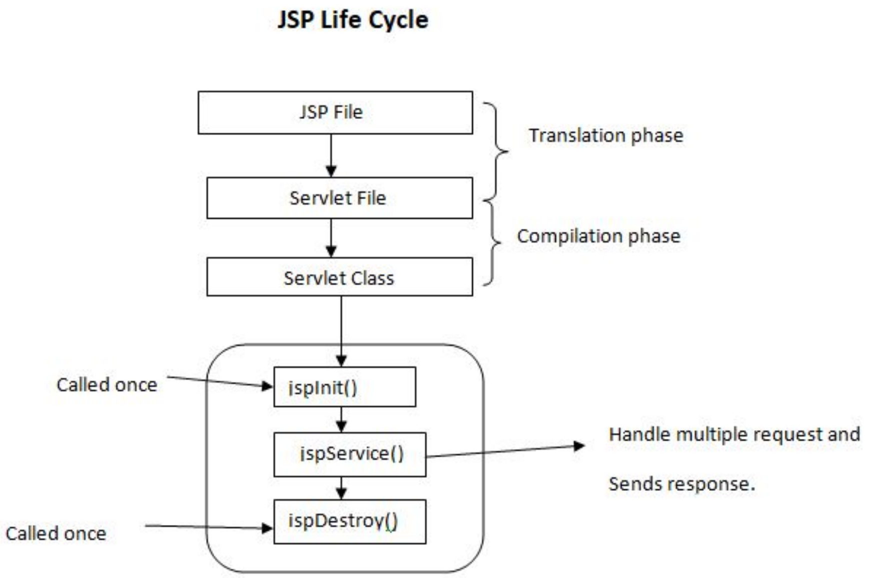
JSP Lifecycle

The JSP life cycle follows this process from its creation to its destruction. Similar to servlet life cycle, it only contains some additional steps to compile the JSP to a servlet.



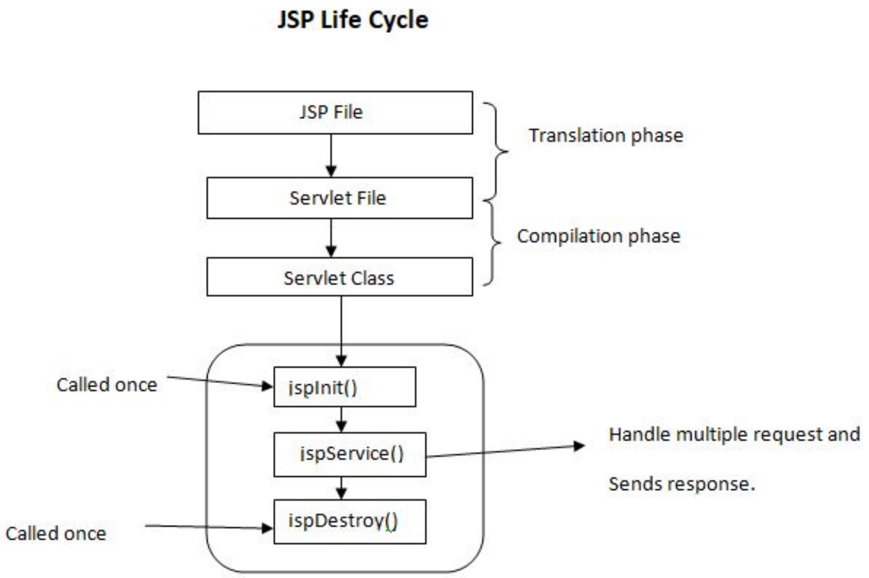
Compilation

- ➔ A browser will request for the JSP
- ➔ JSP engine will check if page needs to be compiled
- ➔ **JSP translated and converted to servlet file** (from a .jsp file to .java file)
- ➔ **Servlet file** (.java file we created) will then be **compiled** (to .class file)



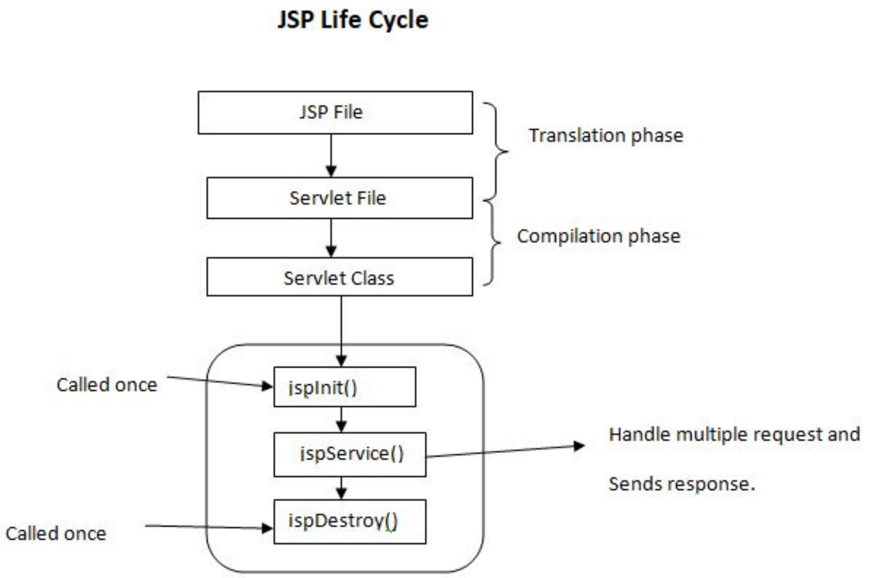
Initialization

- ➔ JSP is loaded in
- ➔ The **jspInit()** method is called
  - ◆ Only done so once during lifecycle
- ➔ Method must be called before completing any requests
- ➔ jspInit() method can be overridden to **initialize anything needed beforehand** (ex: database connections)



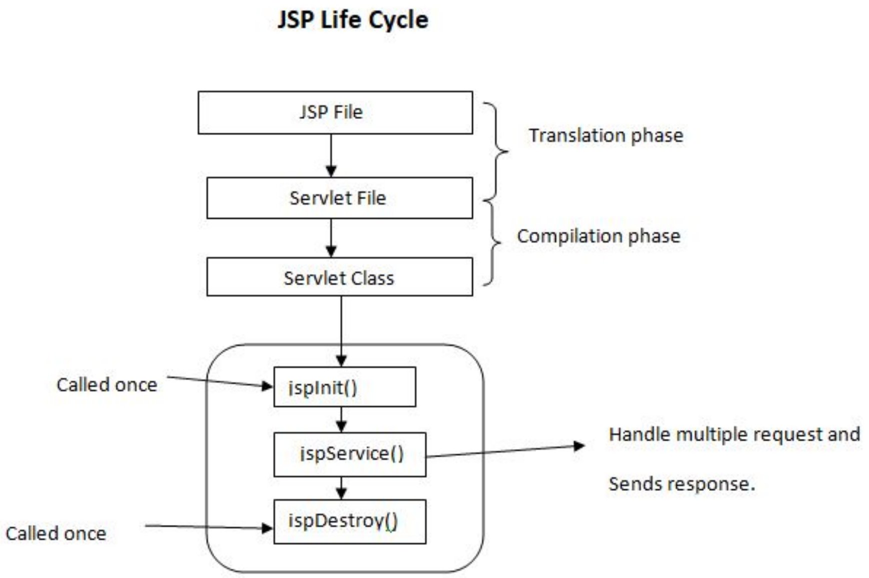
Execution

- ➔ **_jspService()** method used to handle requests for JSP
- ➔ **All requests and interactions with JSP handled here** until it is destroyed

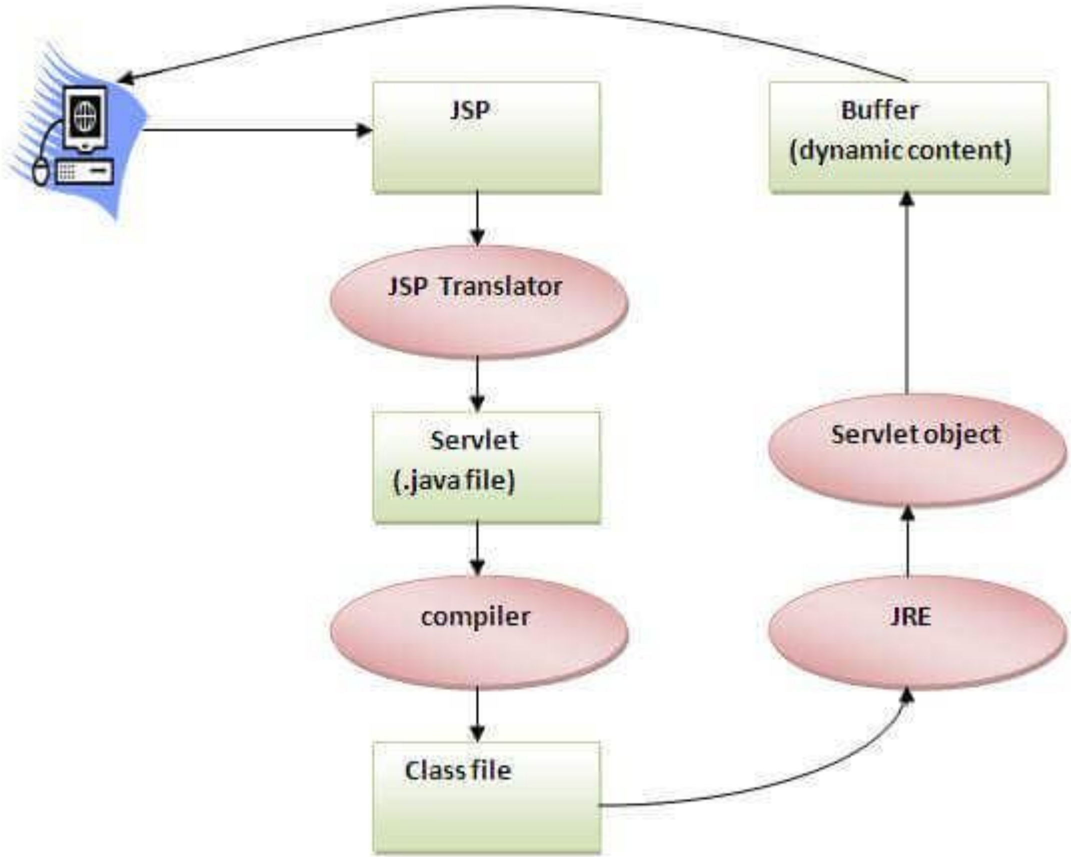


Cleanup

- ➔ ***JSP is removed from container***
- ➔ **jspDestroy()** method called to do any **clean up** needed for removing JSP (just like how the destroy() method in servlets does)



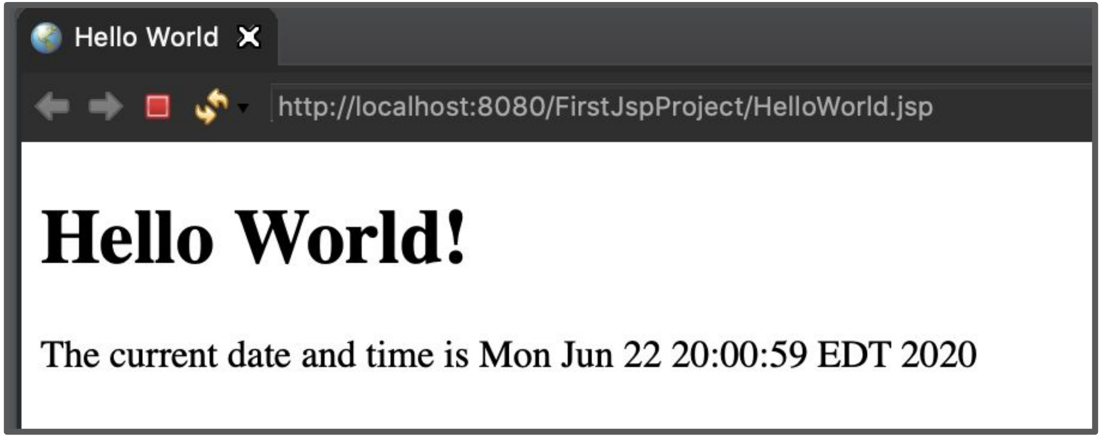
Overview of JSP Lifecycle



Creating JSPs

First JSP Project

- ➔ Create a JSP page in a Dynamic Web Project
- ➔ Display the current date and time using the Date class from java.util package

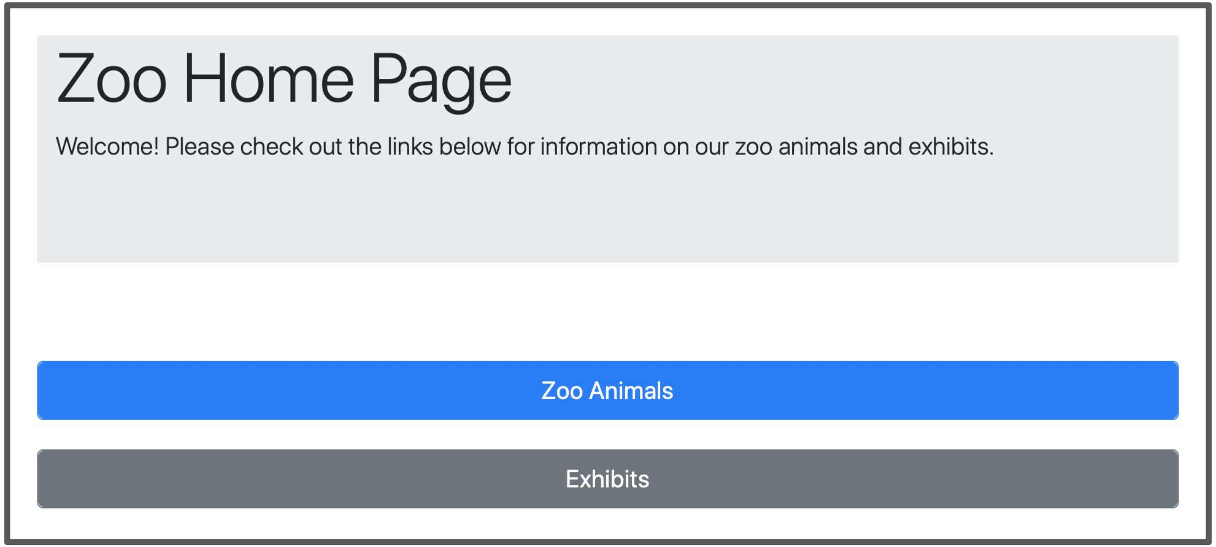


Hello World!

The current date and time is Mon Jun 22 20:00:59 EDT 2020

Zoo and JSTL Project

- Redirect servlet requests to our JSP pages
- Using core JSTL tags to do loops and if statements



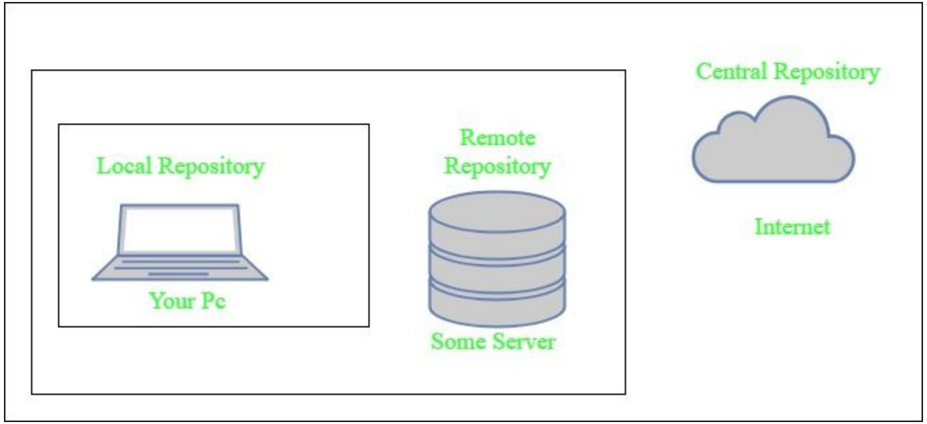
Zoo Home Page

Welcome! Please check out the links below for information on our zoo animals and exhibits.

Maven Project

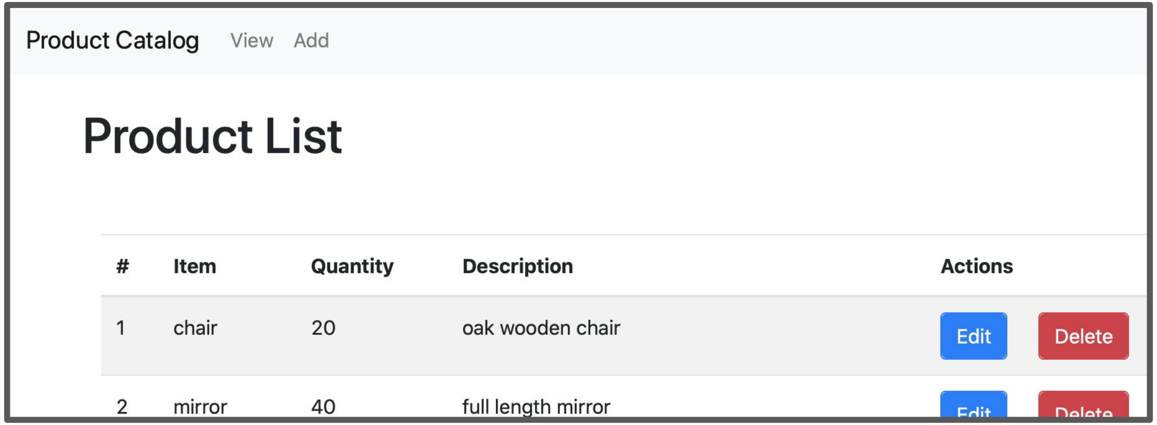
What is a Maven Project?

- ➔ A project type that simplifies the build process
- ➔ Most importantly, it's easy to add jars and other dependencies to our project without additional set up
- ➔ Easy to add pluggins
- ➔ Contains pom.xml file that has configuration information and project operations



CRUD Project

- Create, Read, Update, and Delete from a table
- Use our JSP to view this data
- Send in requests to manipulate this how this data will be displayed and what structure our webpage will take



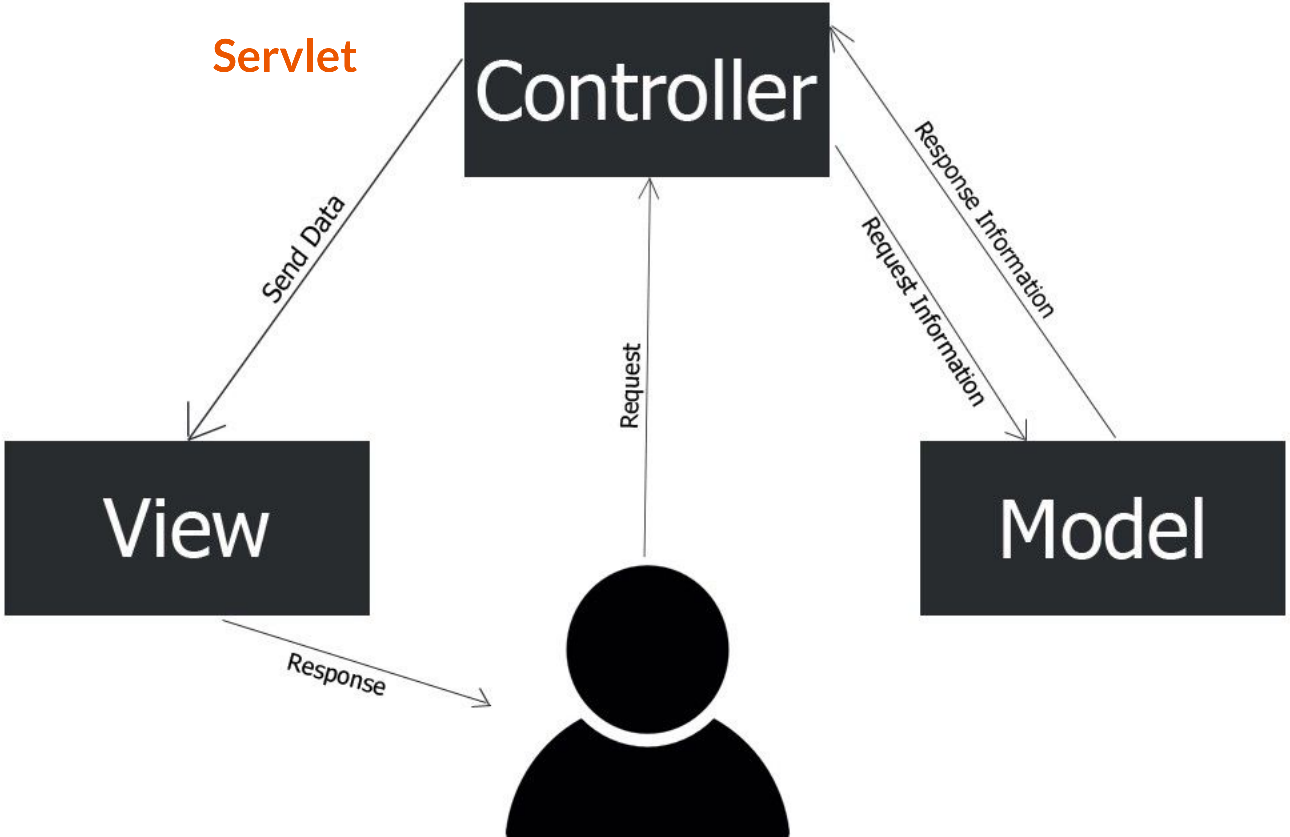
Product List

#	Item	Quantity	Description	Actions
1	chair	20	oak wooden chair	Edit Delete
2	mirror	40	full length mirror	Edit Delete

Model View Controller (MVC)

Servlet

JSP
Pages



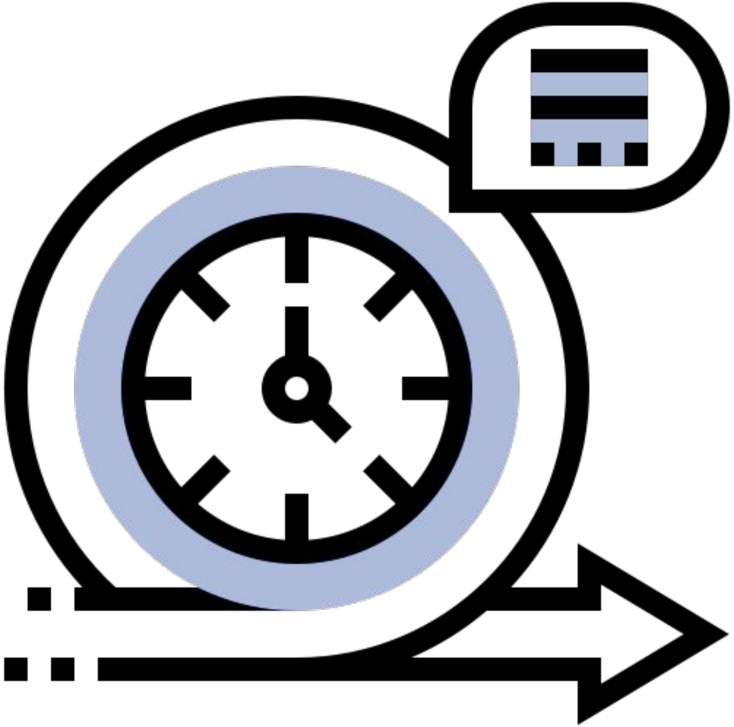
Model Object

Assignment: Library Project

- ➔ Project will use JDBC, Servlets, JSPs, and Maven
- ➔ Create website for a library where you can login as…
  - ◆ A librarian who will manage books and approve patron accounts
  - ◆ A patron who can checkout and return books

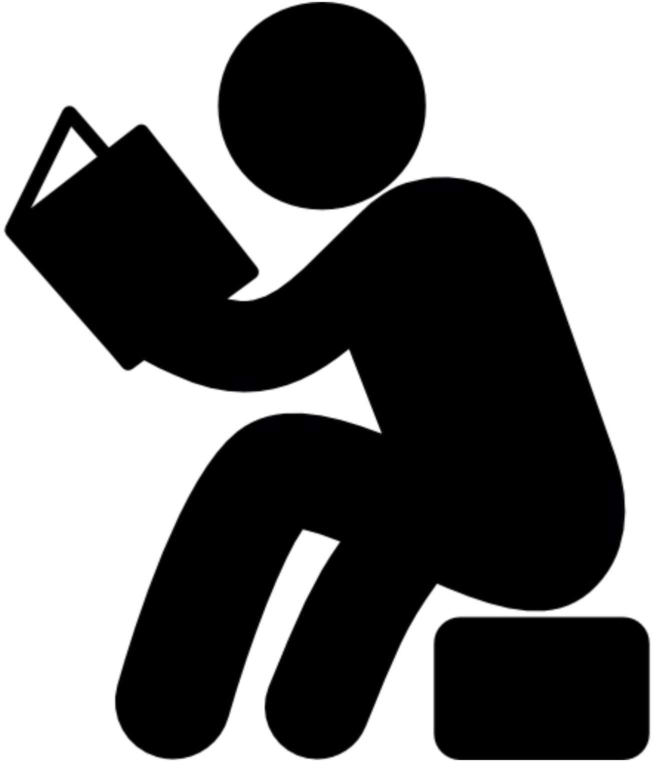
Assignment: Library Project

- ➔ Each team will have a **team lead** and **scrum master**
- ➔ Following Agile…
  - ◆ Create user stories, UMLs, etc. to plan out application
  - ◆ Daily scrum meetings
  - ◆ Create product backlog to track progress (use free apps like Jira)



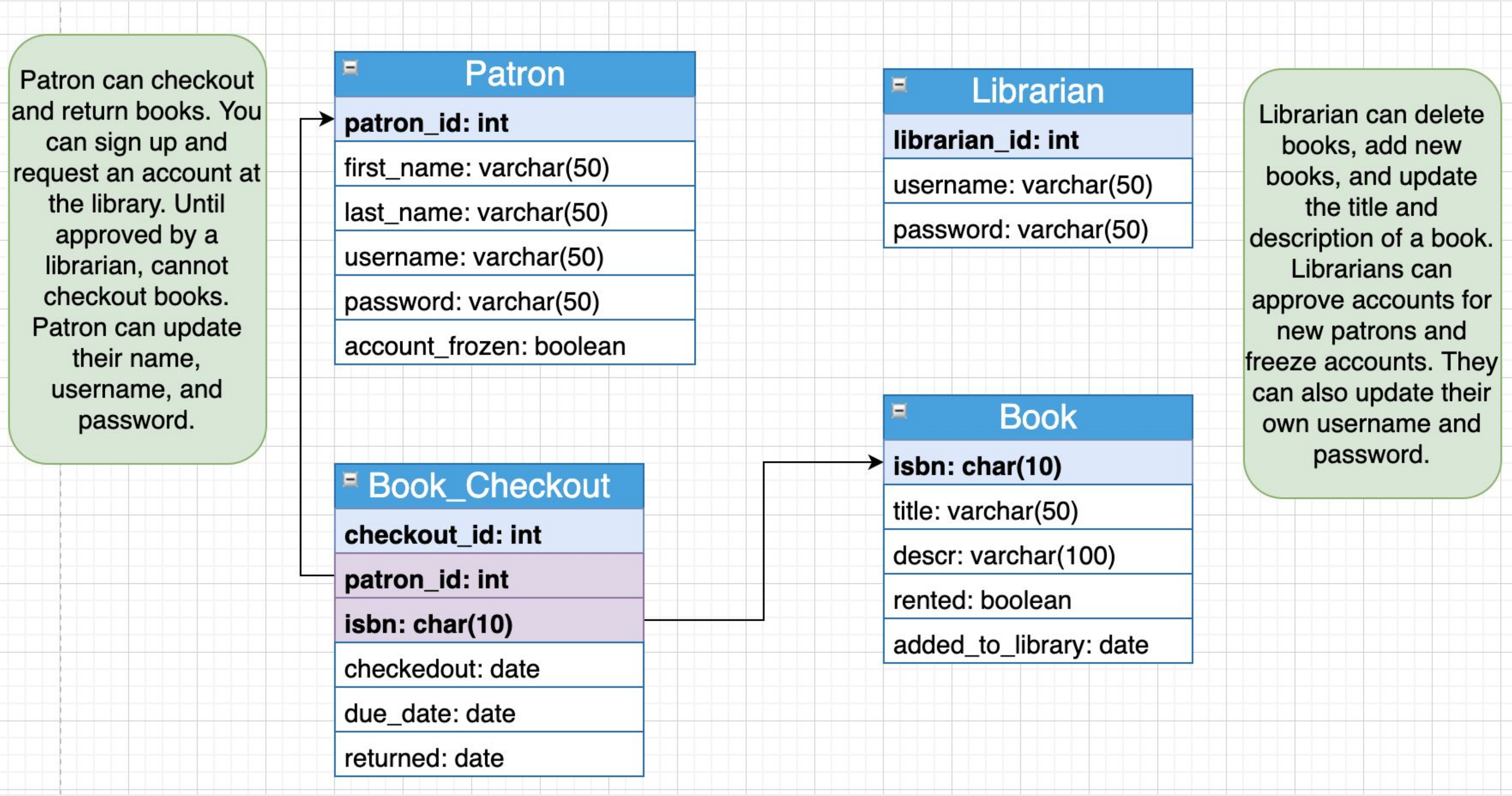
Patron User

- Sign up for an account
  - Pass info for account in a form
  - Account will be “frozen” once created and will not be able to checkout any books
  - Librarian will have to unfreeze account
- Log into their account to…
  - Checkout books (as long as they’re available)
  - Return books
  - View all books previously checkout out as well as current books checkout out
  - List of books at the library
- Update their name, username, and password



Librarian User

- ➔ Add in new books
  - ◆ Must have false set for rented column
  - ◆ Must have today's date for added_to_library column
- ➔ Update only the book's title and description
- ➔ Approve accounts for patrons
  - ◆ Change their accounts from frozen to unfrozen so they can checkout books
- ➔ Update their username and password



Patron can checkout and return books. You can sign up and request an account at the library. Until approved by a librarian, cannot checkout books. Patron can update their name, username, and password.

Patron
patron_id: int
first_name: varchar(50)
last_name: varchar(50)
username: varchar(50)
password: varchar(50)
account_frozen: boolean

Librarian
librarian_id: int
username: varchar(50)
password: varchar(50)

Librarian can delete books, add new books, and update the title and description of a book. Librarians can approve accounts for new patrons and freeze accounts. They can also update their own username and password.

Book_Checkout
checkout_id: int
patron_id: int
isbn: char(10)
checkedout: date
due_date: date
returned: date

Book
isbn: char(10)
title: varchar(50)
descr: varchar(100)
rented: boolean
added_to_library: date

FIN