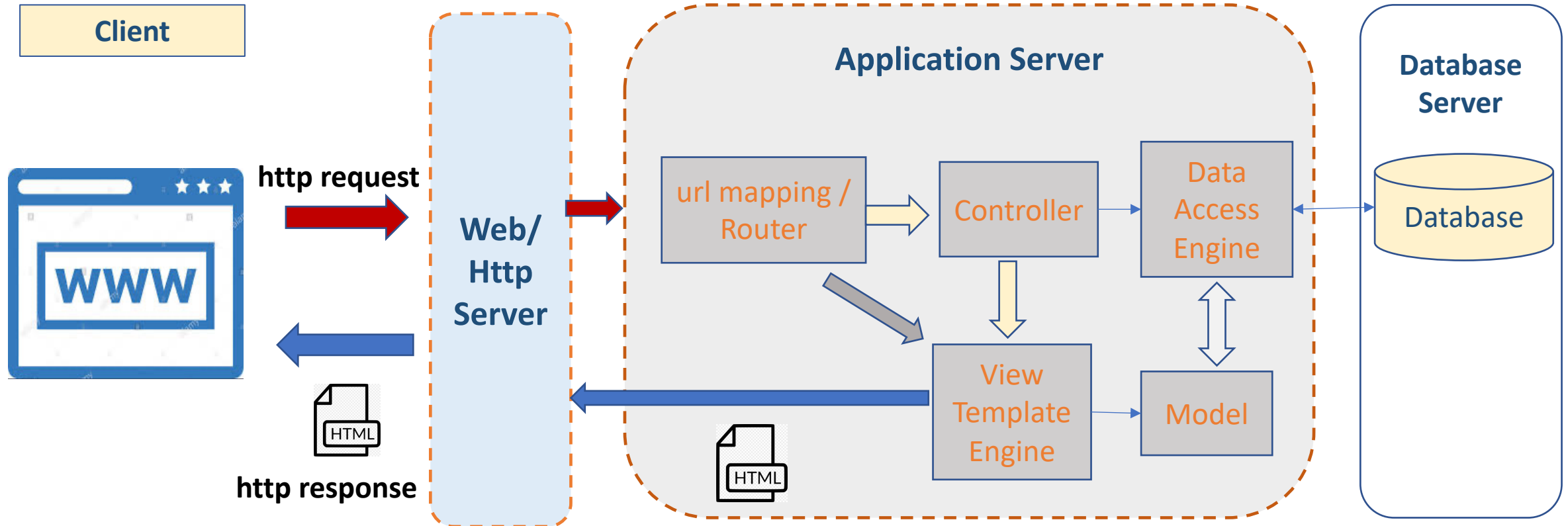


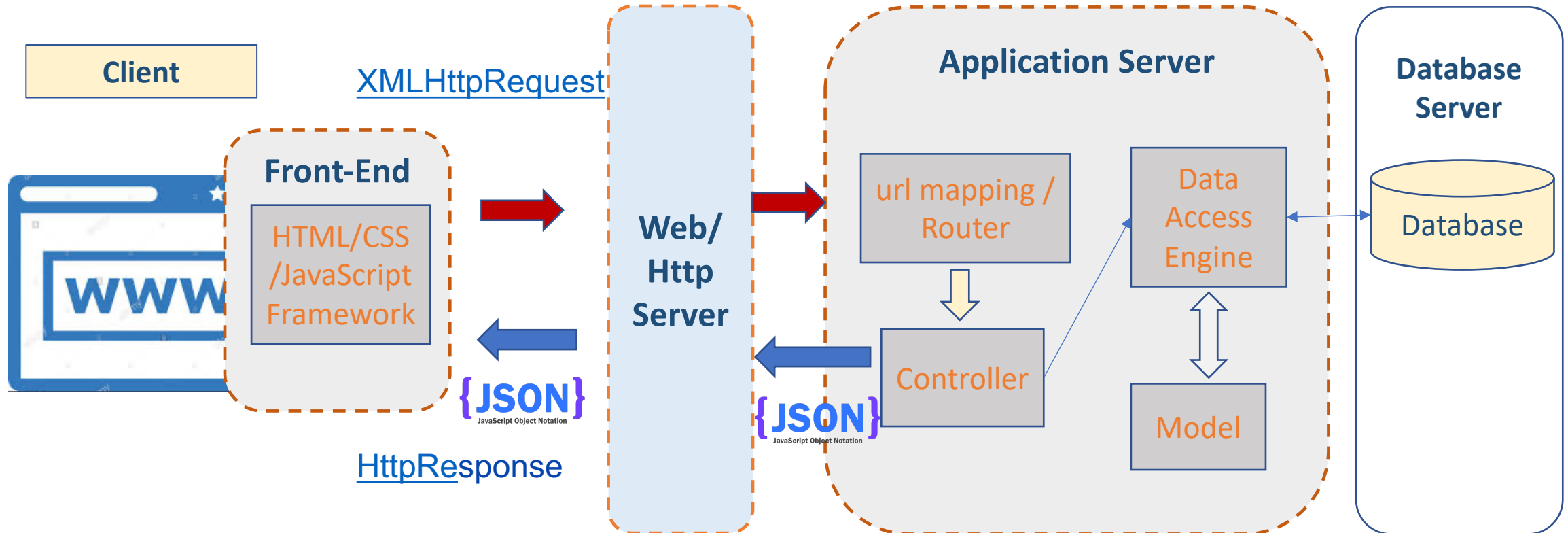
INT202 Server-Side Web Programming I (1-2-4)

pichet@sit.kmutt.ac.th

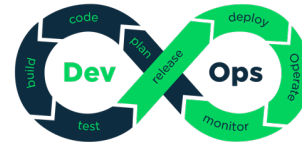
Multiple Pages (Classic) Web Application



MVC Web Application Architectures (modern/spa)



Full Stack Developer



Client Side

Server Side

{ REST }

NoSQL

SQL

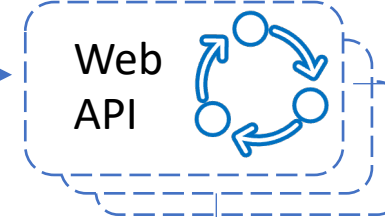
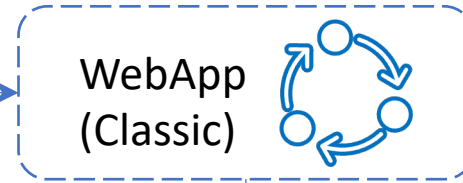
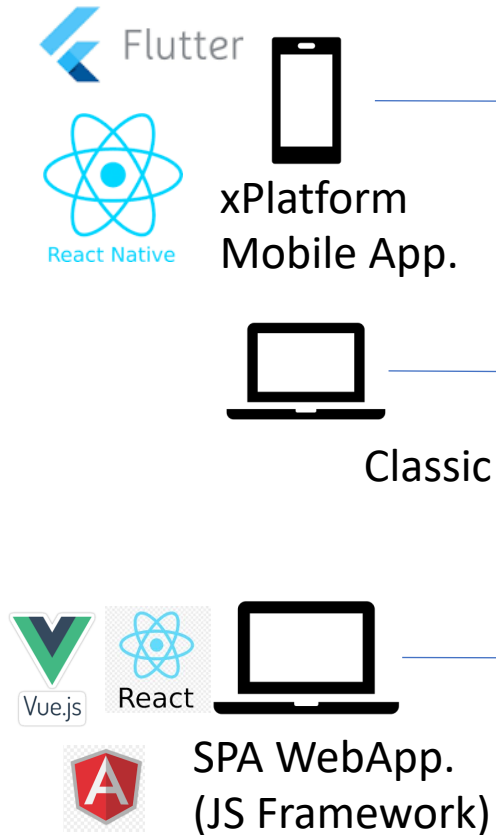
Microservice

Web API

Database

EIS: Enterprise Info. System

Database



Full Stack Web Developer

- A full stack web developer is a person who can develop both client and server software.
 - Program a browser (like using JavaScript, jQuery, Angular, or Vue)
 - Program a server (like using JavaEE, PHP, ASP, Python, or NodeJS)
 - Program a database (like using SQL, SQLite, or MongoDB)



https://www.w3schools.com/whatis/whatis_fullstack.asp

Web Development Roadmaps (1)

- Front-End Roadmap

- Every Web Developer must have a basic understanding of HTML, CSS, and JavaScript.
- Responsive Web Design is used in all types of modern web development.
- ECMAScript 5 (JavaScript 5) is supported in all modern browsers.
- On the CSS side you should choose a framework for responsive web design:
 - Bootstrap / Material Design / W3.CSS / Tailwind
- On the JavaScript side you should learn at least one modern framework:
 - React.js / Angular.js / **Vue.js** / W3.JS



<https://www.w3schools.com/whatis/default.asp>

Web Development Roadmaps (2)

- Back-End Roadmaps
 - SQL / NoSQL
 - Language: Java / PHP / ASP, C# / Python / Node.js
 - Frame work: Spring Boot / Laravel / .NET / Django / Adonis, Express
 - Web Service
 - Microservice

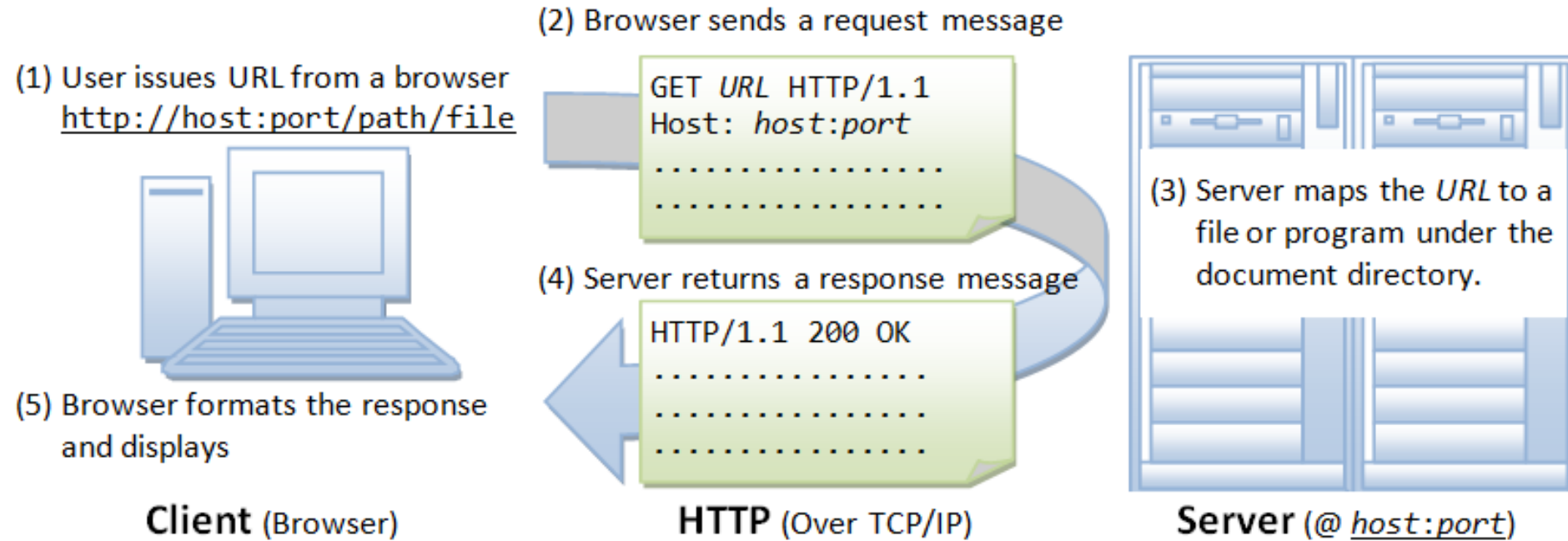


https://www.w3schools.com/whatis/whatis_fullstack.asp

World Wide Web Communication

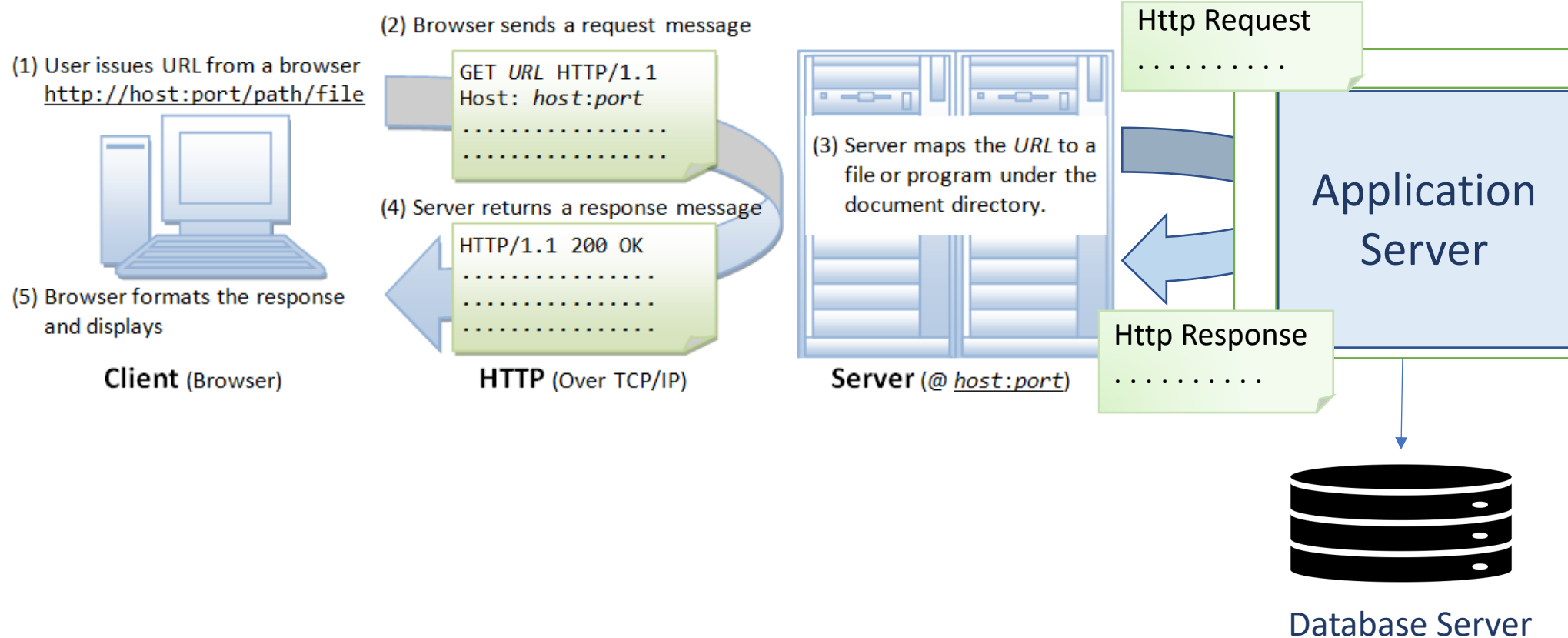
- **HTTP** stands for **H**yper **T**ext **T**ransfer **P**rotocol
- **WWW** is about communication between web **clients** and **servers**
- Communication between client computers and web servers is done by sending **HTTP Requests** and receiving **HTTP Responses**
- **Clients** are often browsers (Chrome, Edge, Safari), but they can be any type of program or device.
- **Servers** are most often computers in the cloud.

HTTP Protocol

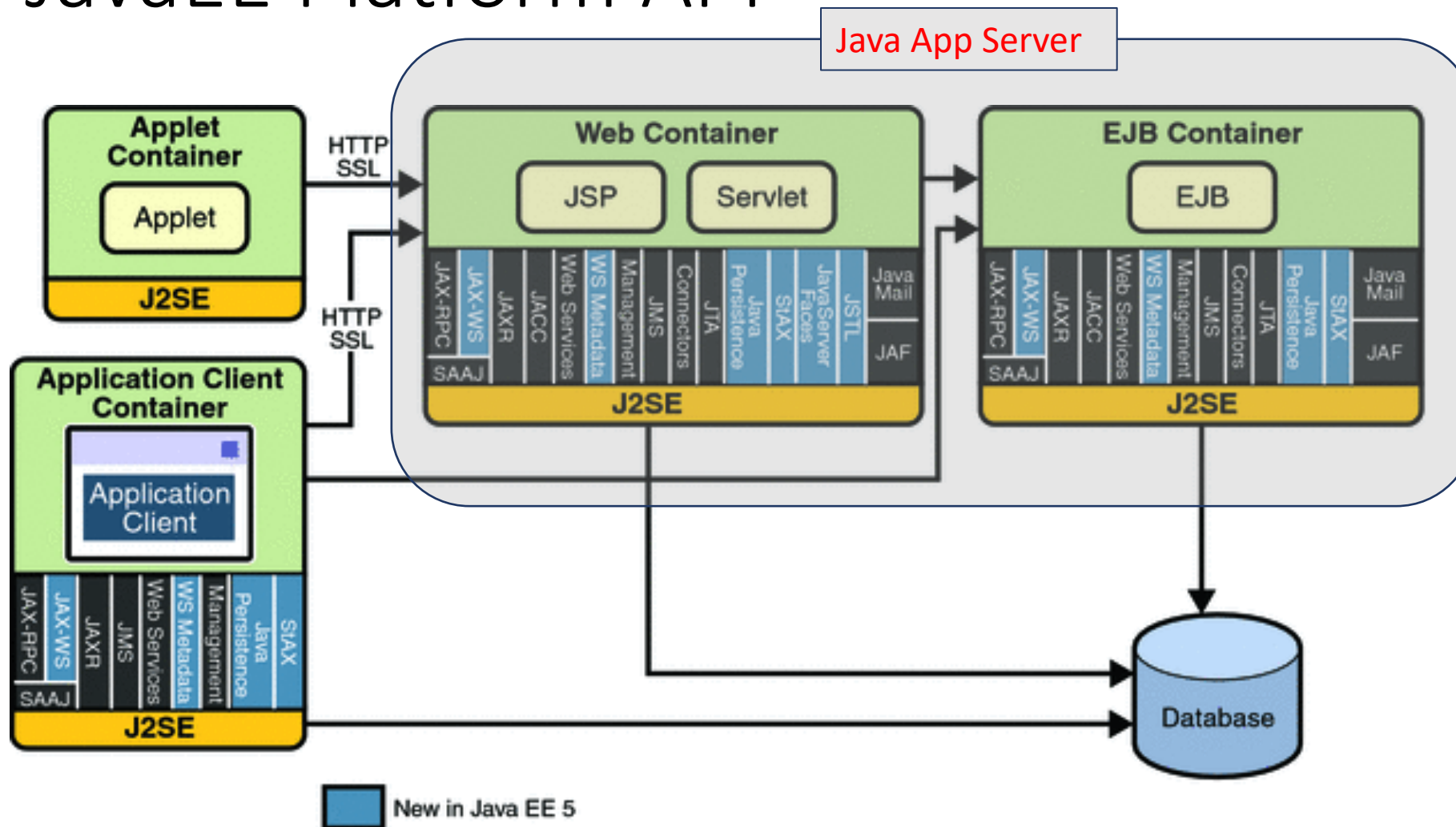


https://www.w3schools.com/whatis/whatis_http.asp

Web Application



JavaEE Platform API



<https://docs.oracle.com/javaee/5/tutorial/doc/bnacj.html>



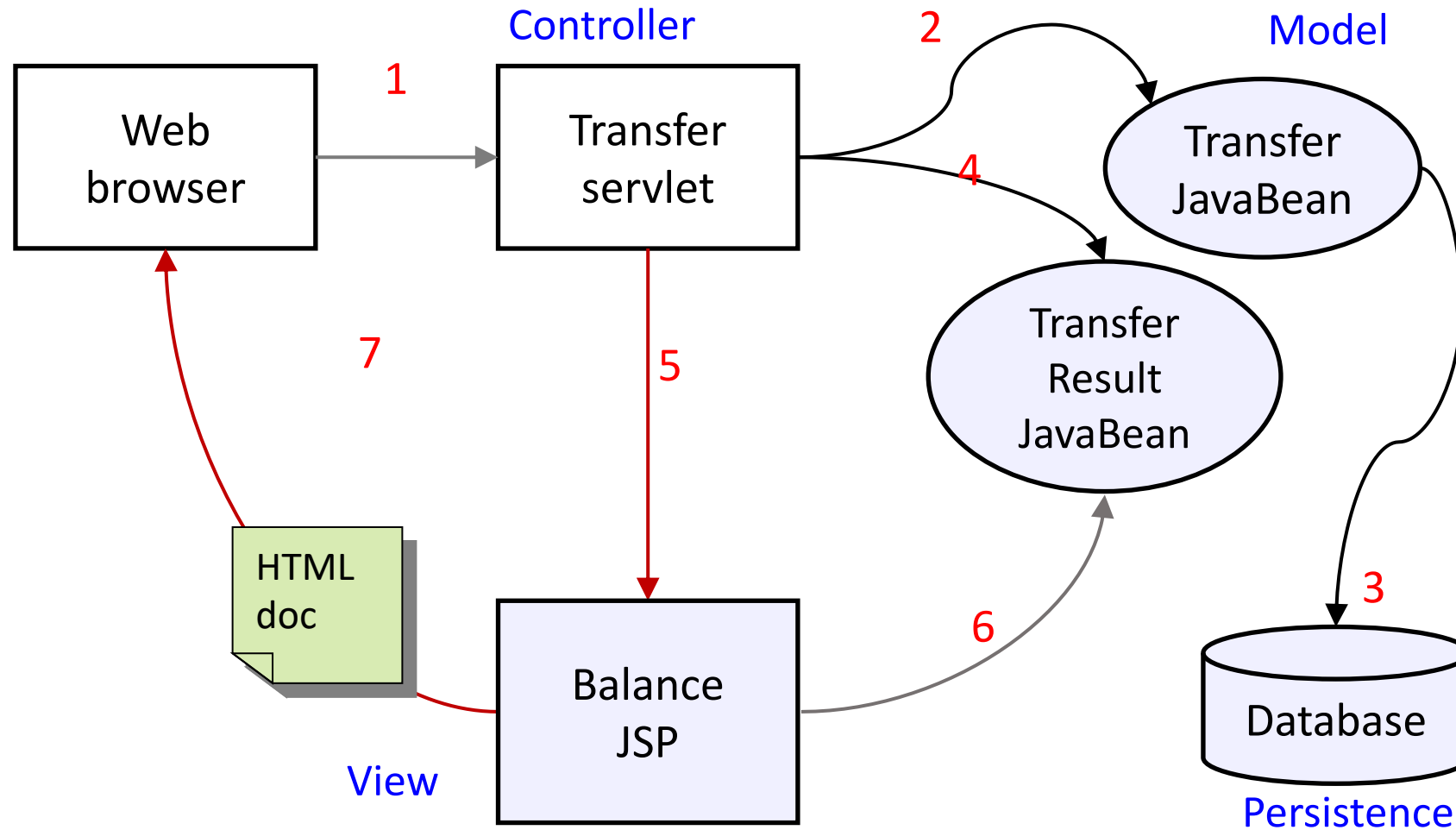
Servlet Example

```
package com.ibm.example.servlet;
import javax.servlet.http.HttpServlet;
...
public class VerySimpleServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        String browser = request.getHeader("User-Agent");
        response.setStatus(HttpServletResponse.SC_OK);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML><HEAD><TITLE>Simple servlet");
        out.println("</TITLE></HEAD><BODY>");
        out.println ("Browser details: " + browser);
        out.println("</BODY></HTML>");
    }
}
```

MVC: application to Java EE

Client

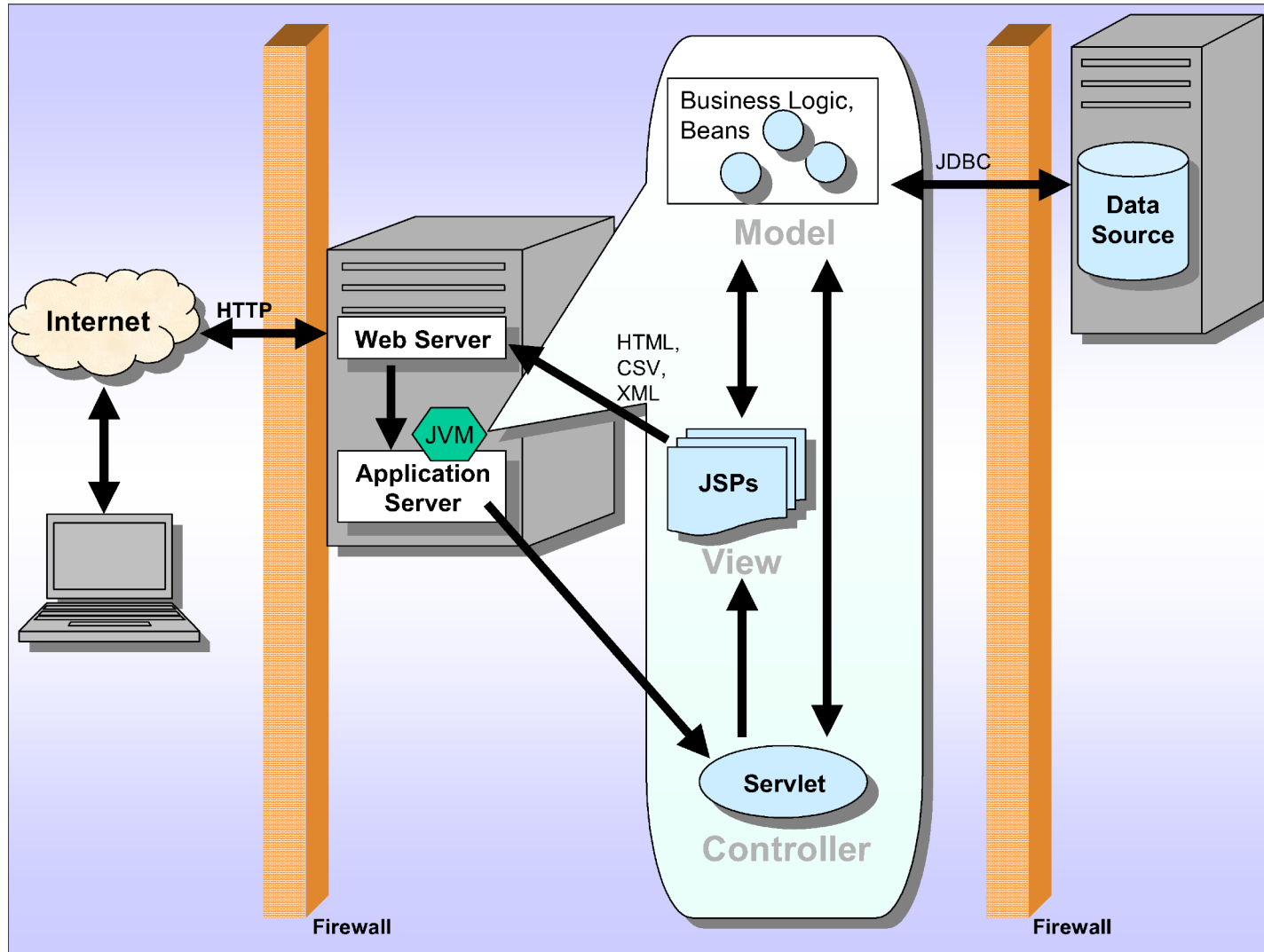
Java EE server



MVC: benefits

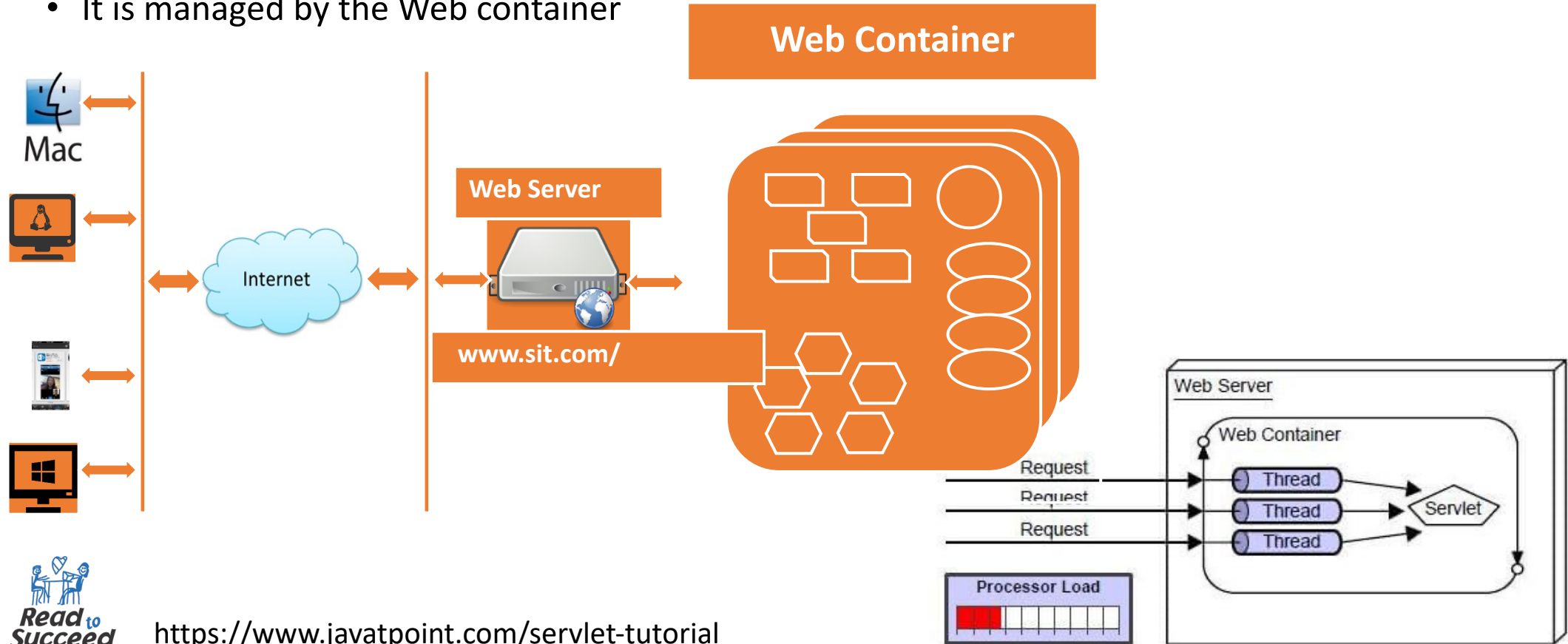
- Promotes code reuse
 - The purpose of the model is to provide business logic and data access in one place
 - You can reuse this logic in many applications at the same time, without the need for any extra coding
- Reduces development time
 - The model, view, and controller are developed in parallel
- Is more maintainable
 - You can change the view without affecting the model
 - You can change the Web page view to display a chart instead of a table with no change to the model
 - You can change the model without affecting the view
 - For example, the way in which an insurance premium is calculated changes, but the interface to the business method remains the same
 - You can move data without affecting the view or model
 - The layering concept allows for more flexibility

Java Web Application Development (MVC)



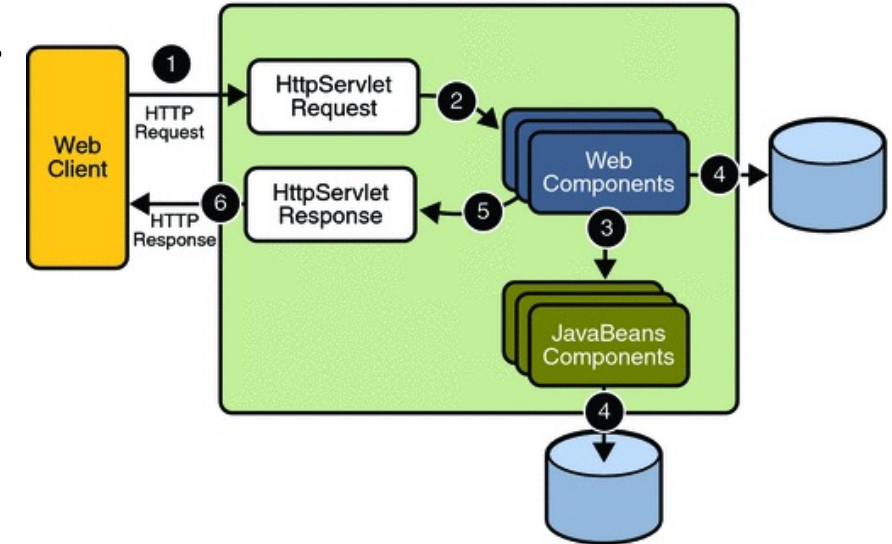
What is a servlet?

- A servlet is a standard, server-side component of a Java EE application that executes business logic on behalf of an HTTP request
 - It runs in the server tier (and not in the client)
 - It is managed by the Web container



Servlet process flow

- The client makes a request naming a servlet as part of the URL.
- The Web server forwards the request to a Web container, which locates an instance of a servlet class.
- The Web container calls the servlet's service method.
- The servlet builds a response dynamically, and passes it to the Web server. External resources may also be used.
- The Web server sends the response back to the client.



Request protocol

- The request object encapsulates all of the information from the client request. The following methods are available to access parameters:
 - **String getParameter(String name)**

```
String name = request.getParameter("username");
```
 - **Enumeration getParameterNames()**

```
Enumeration<String> params = request.getParameterNames();  
while(params.hasMoreElements()) {  
    String name = params.nextElement();  
}
```
 - **String[] getParameterValues(String name)**

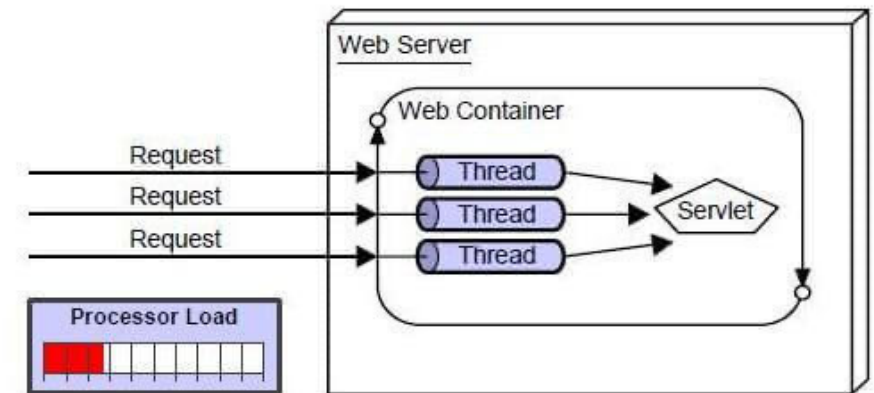
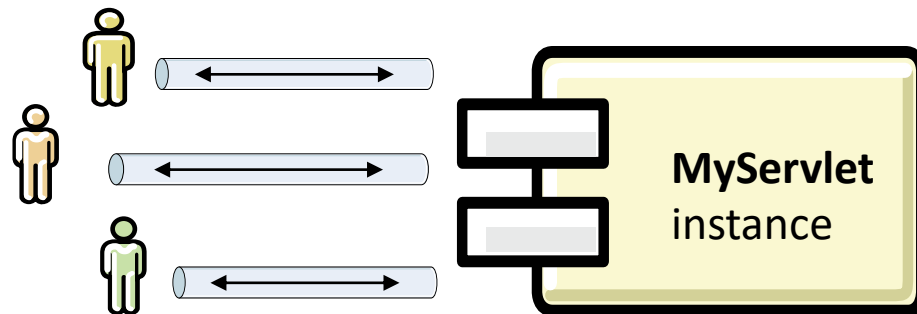
```
String favoriteSubj[] =  
request.getParameterValue("subjects");
```
 - **Map<String,String[]> getParameterMap()**
 - **Map<String,String> paramMap = request.getParameterMap();**

Processing input data example

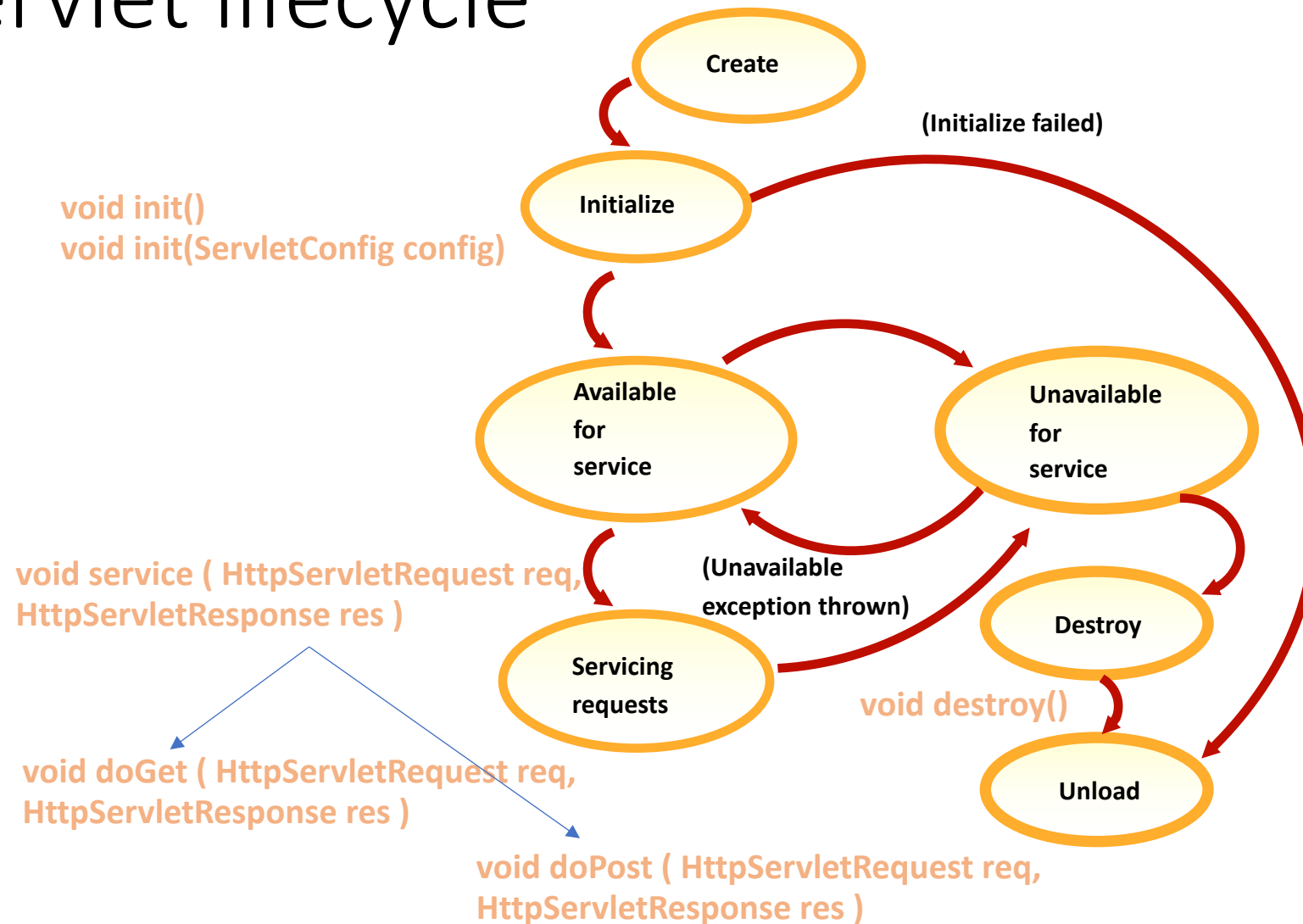
```
public void processRequest(HttpServletRequest request,
                          HttpServletResponse response) throws... {
    String name = request.getParameter("name");
    if (name==null) {
        name = "unknown";
    } else if (name.length== 0) {
        name = "missing";
    }
    String height[] = request.getParameterValues("height");
    if (height == null) {
        height = new String[] {"unknown"};
    }
    for (int i = 0; i < height.length; i++){
        if (height[i].length== 0) height[i] = "missing";
    }
}
```

Building a simple Java servlet

- To create a servlet that responds to HTTP requests, you must:
 - Create a class that extends `javax.servlet.http.HttpServlet`
 - Override the `doGet` or `doPost` methods to process HTTP GET or POST requests
 - Process `HttpServletRequest` input values
 - Invoke the business process
 - Set the `HttpServletResponse` values
 - Output HTML to the output `PrintWriter`
- Servlets are inherently multithreaded



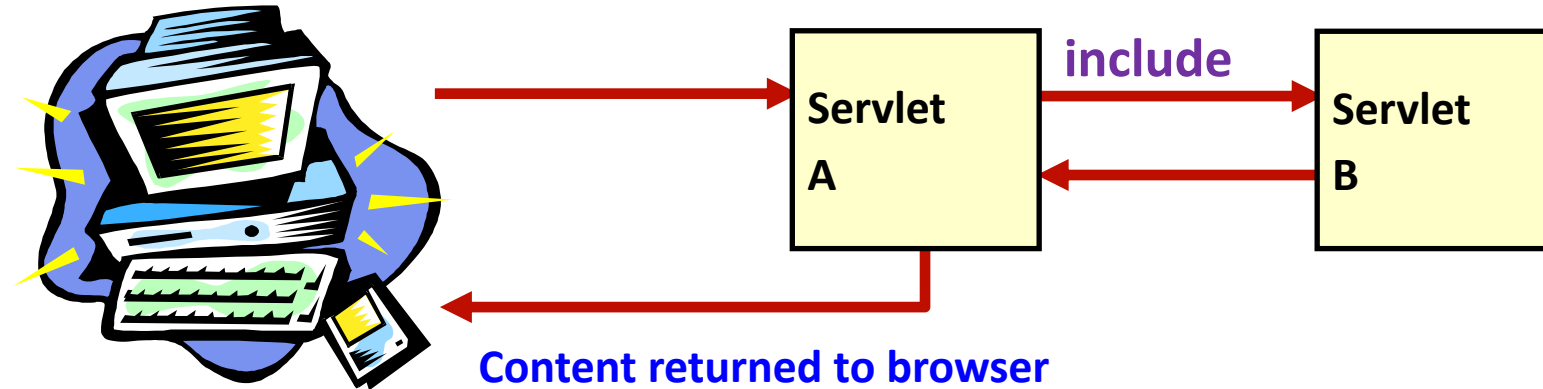
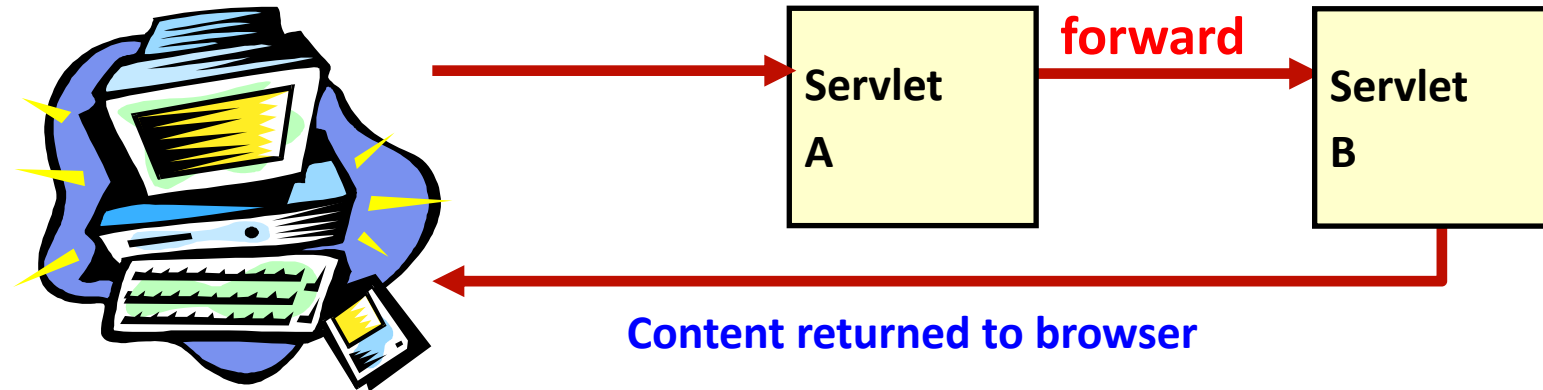
Java servlet lifecycle



Request dispatcher

- The RequestDispatcher allows you to forward a request to another servlet, or to include the output from another servlet
- Used to support both forwarding processing to, and including response from, a variety of local Web resources
 - For example, JSP pages and HTML files
- If a reference to the RequestDispatcher is acquired from the ServletContext
 - Path information is relative to the ServletContext
- If a reference to the RequestDispatcher is acquired from the HttpServletRequest
 - Path information is relative to the path of the current request

Request dispatcher flow

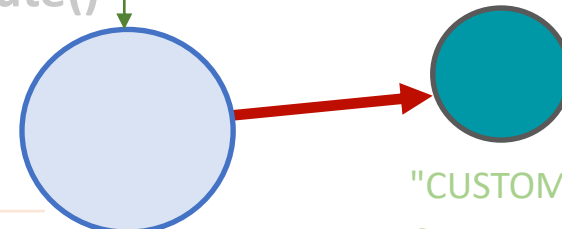


Sharing objects example

```
// Servlet "A"  
public void doGet (HttpServletRequest request, HttpServletResponse resp)... {  
    // process request headers & query data  
    Customer cust;  
    ...  
    request.setAttribute("CUSTOMER", cust);  
    String viewPage = "/CustomerInfo.jsp";  
    getServletContext().getRequestDispatcher(viewPage).forward(request, resp);  
}
```

1

setAttribute()



"CUSTOMER"
Customer

2

getAttribute()

HttpServletRequest

```
// CustomerInfo.jsp  
<%  
    Customer aCust = (Customer) request.getAttribute("CUSTOMER");  
>%
```


JSP :

JavaServer Pages

JakartaServer Pages



<https://www.javatpoint.com/jsp-tutorial>

What is JakartaServer Pages technology?

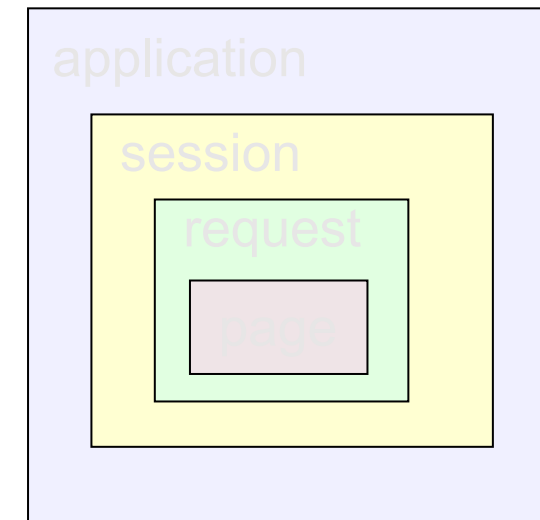
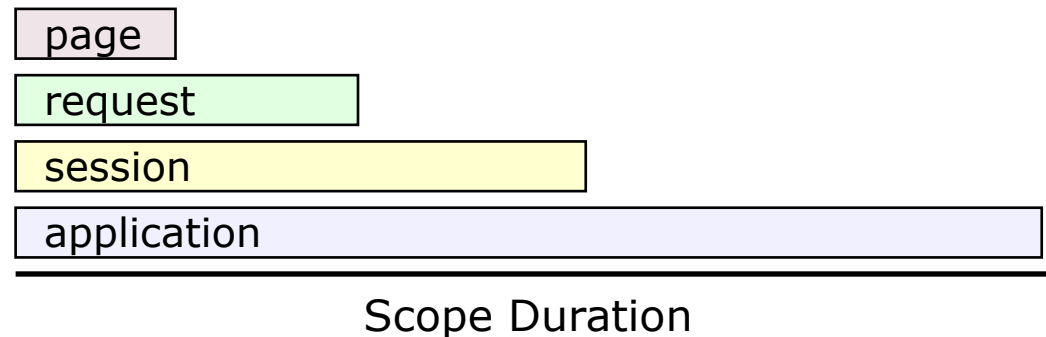
- JakartaServer Pages technology that lets you mix static HTML with dynamically-generated HTML
- JSP technology allows server-side scripting:
 - Static tags are HTML, XML, or another markup language.
 - Dynamic content generated by scripting code
 - Java is the (default) scripting language
- A JSP file (with an extension of .jsp) contains any combination of:
 - JSP syntax
 - Markup tags such as HTML or XML

JSP execution model

- A JSP page is executed in a Web container
 - The Web container delivers client requests to the JSP page, and returns the page's response to the client
- The JSP page is converted into a servlet (JSP servlet) and executed
- This process is known as page compilation:
 - JSP source is parsed
 - Java servlet code is generated
 - This JSP servlet is compiled, loaded, and run

Scope attributes

- A JSP page can access objects at run time via one of four different scopes (or holder objects)
 - Page
 - The current JSP page, used with Custom Actions
 - Request
 - The current HttpServletRequest object
 - Session
 - The current HttpSession object
 - Application
 - The current ServletContext object



JSP syntax elements

- JSP elements fall into four groups:
 - Directives
 - ~~• Scripting
 - Declarations
 - Expressions
 - Scriptlets~~
 - Comments
 - Actions

JSP directives

- JSP directives are instructions processed by the JSP engine when the page is compiled into a servlet
 - `<%@ directive {attribute="value"}* %>`

- JSP technology currently defines page, include, and taglib directives

```
<%@ page language="java" %>
```

```
<%@ include file="companyBanner.html"%>
```

```
<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>
```

JSP EL

JSP Expression Language

What is JSP EL?

- JSP Expression Language (EL) is a simple language based on:
 - Local variables of the current page
 - Nested properties and accessors to collections
 - Relational, logical and arithmetic operators
 - A set of implicit objects
 - Extensible functions mapping into static methods in Java classes
- EL is invoked via the construct `${ ... }`
- EL may be used:
 - In attribute values for standard and custom actions
 - Within template text

EL Examples

- EL can be used directly in template text
 - `<h2>`
 - Hello, `${user.firstName}` `${user.lastName}`
 - `</h2>`
- EL expressions can be used in any attribute that can accept a run-time expression
 - `<c:if test="${a<3}">`
 - ...
 - `</c:if>`

Implicit Objects

- Several implicit objects are available to EL expressions used in JSP pages:
 - pageContext
 - pageScope
 - requestScope
 - sessionScope
 - applicationScope
 - param
 - paramValues
 - header
 - headerValues
 - cookie
 - initParam

Syntax Overview

- Variables are accessed by name
- Generalized [] and . operators
 - Can be used to access
 - Maps
 - Lists
 - Arrays of objects
 - JavaBeans properties
 - Can be nested arbitrarily
- Relational comparisons
- Arithmetic and Logical operators
- Example: to access the name property of the user
- bean in session scope:
 - `${sessionScope.userName}`

Basic Syntax Elements

- Literals
 - boolean, integer, floating point, string and null
- Operators
 - Accessor
 - . []
 - Arithmetic
 - + - * / %
 - Relational
 - == != < > <= >=
 - Logical
 - && || !
 - Empty
 - empty
 - Conditional
 - ? :

```
${param.cost * 1.085}
```

```
${empty sessionScope ? "yes" : "no"}
```

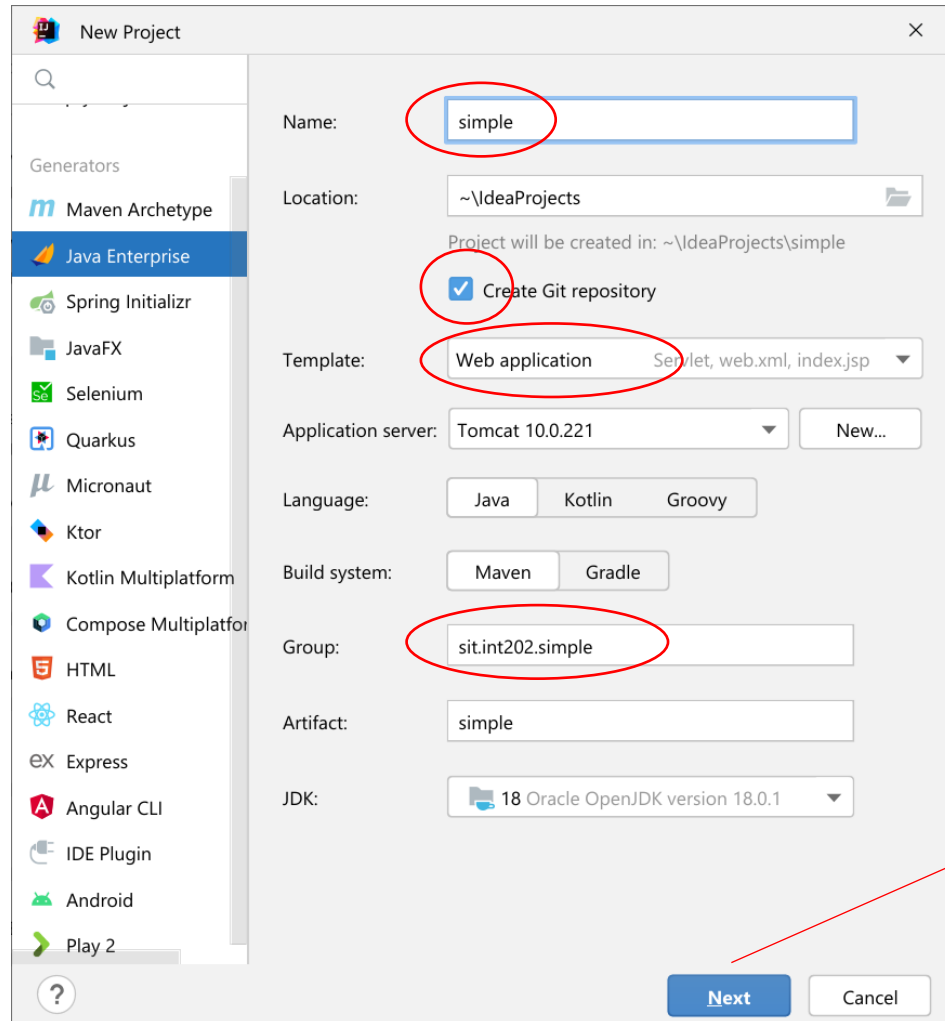
Named Variables

- EL contains a variable resolver which will search the page, request, session, and application scopes for values
 - Example: `${user}` will resolve to `sessionScope.user`
- If the attribute is not found in any scope, null is returned
- The specification is vague as to what happens when likenamed attributes are found in more than one scope

INT202: Nano Lab

Create/Run-Maven Web App Project

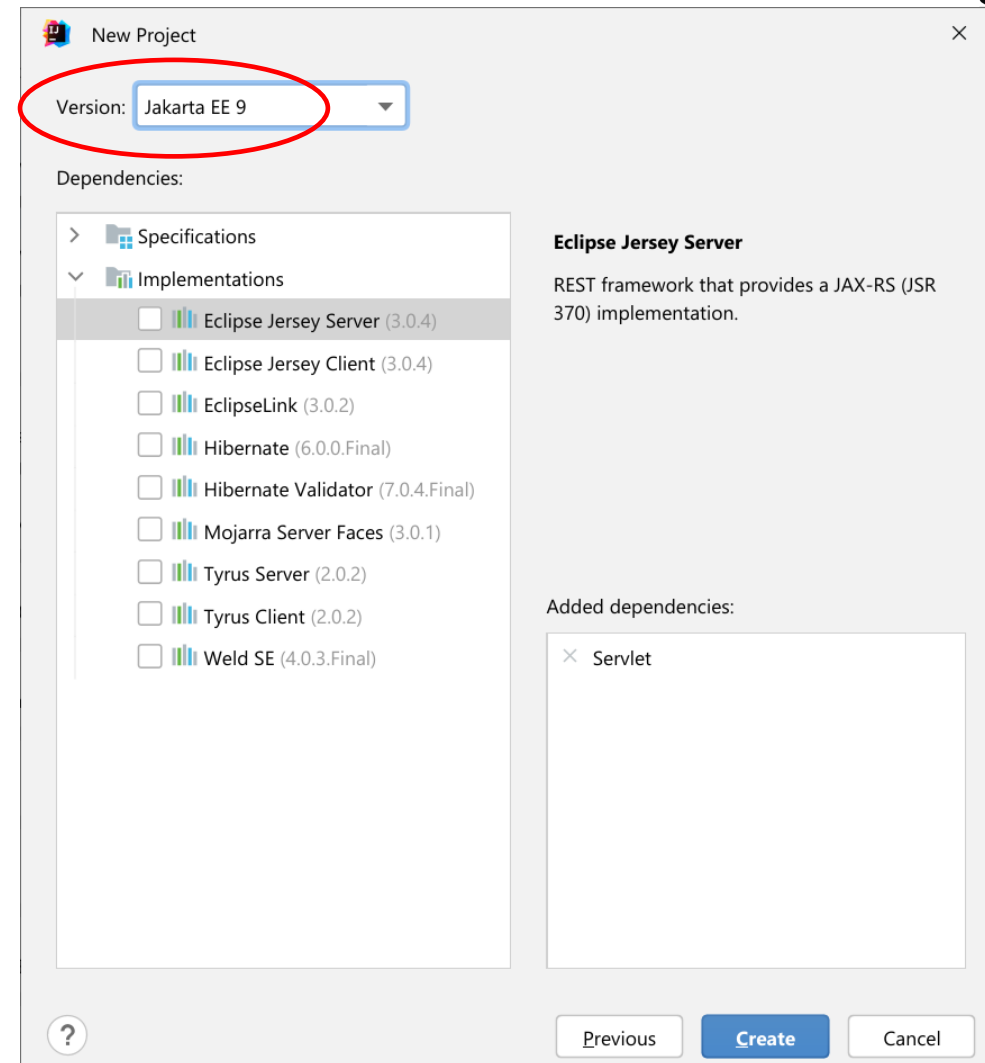
Create Maven Project with IntelliJ Idea



The first dialog shows the 'New Project' wizard. The 'Generators' list on the left has 'Java Enterprise' selected. The main form contains the following fields and options:

- Name:** simple
- Location:** ~\IdeaProjects
- Project will be created in:** ~\IdeaProjects\simple
- ☒ **Create Git repository**
- Template:** Web application
- Application server:** Tomcat 10.0.221
- Language:** Java, Kotlin, Groovy
- Build system:** Maven, Gradle
- Group:** sit.int202.simple
- Artifact:** simple
- JDK:** 18 Oracle OpenJDK version 18.0.1

At the bottom, there are 'Next' and 'Cancel' buttons. A red arrow points from the 'Next' button to the second dialog.



The second dialog shows the 'New Project' wizard, Step 2. The 'Version' dropdown is set to 'Jakarta EE 9'. The 'Dependencies' section shows a list of dependencies under 'Implementations':

- ☐ Eclipse Jersey Server (3.0.4)
- ☐ Eclipse Jersey Client (3.0.4)
- ☐ EclipseLink (3.0.2)
- ☐ Hibernate (6.0.0.Final)
- ☐ Hibernate Validator (7.0.4.Final)
- ☐ Mojarra Server Faces (3.0.1)
- ☐ Tyrus Server (2.0.2)
- ☐ Tyrus Client (2.0.2)
- ☐ Weld SE (4.0.3.Final)

On the right, there is a description for 'Eclipse Jersey Server' and an 'Added dependencies' section showing 'Servlet'.

At the bottom, there are 'Previous', 'Create', and 'Cancel' buttons.

The image shows a screenshot of an IDE (IntelliJ IDEA) with a project named 'simple'. The project structure is visible in the left sidebar, showing a 'src' directory containing a 'main' directory, which in turn contains a 'java' directory. The 'java' directory contains several sub-directories: 'sit', 'sit.int202', 'sit.int202.simple', and 'sit.int202.simple.simple'. The 'sit.int202.simple.simple' directory contains a 'HelloServlet' class. The 'pom.xml' file is open in the editor, showing the following XML content:

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <groupId>com.example</groupId>
  <artifactId>simple</artifactId>
  <version>1.0.0</version>
  <packaging>war</packaging>
  <name>simple</name>
  <dependencies>
    <dependency>
      <groupId>org.apache.tomcat</groupId>
      <artifactId>tomcat-servlet-api</artifactId>
      <version>10.0.221</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.tomcat</groupId>
        <artifactId>tomcat-maven-plugin</artifactId>
        <version>2.2</version>
      </plugin>
    </plugins>
  </build>
  <profiles>
    <profile>
      <id>tomcat</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <build>
        <plugins>
          <plugin>
            <groupId>org.apache.tomcat</groupId>
            <artifactId>tomcat-maven-plugin</artifactId>
            <version>2.2</version>
          </plugin>
        </plugins>
      </build>
    </profile>
  </profiles>
  <properties>
    <tomcat.version>10.0.221</tomcat.version>
  </properties>
</project>
```

The 'Tomcat 10.0.221' dropdown menu in the top toolbar is highlighted with a red box. The 'src' directory in the left sidebar is also highlighted with a red box. The 'Services' tab at the bottom shows the 'Tomcat Server' configuration, with the 'Tomcat 10.0.221 [local]' instance listed. The 'simple:war exploded' directory is also visible under the 'Tomcat 10.0.221 [local]' instance.

A web browser window titled 'JSP - Hello World' is shown in the foreground, displaying the output of the JSP page: 'Hello World!' and a link to 'Hello Servlet'.

INT202: Simply MVC Lab

Model View Controller Example

Features: Subject Listing

I. Create Model:

- Subject
- SubjectRepository
- Create Servlet (Controller)
 - Create collection of subject when servlet instantiate
 - Forward collection to JSP for each HTTP Request.
- Create JSP (View)
 - Generate HTTP Response and send back to each client.

```
<dependency>
  <groupId>jakarta.servlet.jsp.jstl</groupId>
  <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
  <version>2.0.0</version>
</dependency>
<dependency>
  <groupId>org.glassfish.web</groupId>
  <artifactId>jakarta.servlet.jsp.jstl</artifactId>
  <version>2.0.0</version>
</dependency>
```

Subject class

```
package sit.int202.simple.entities;  
  
import lombok.*;  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@EqualsAndHashCode  
public class Subject {  
    private String id;  
    private String title;  
    private double credit;  
}
```

SubjectRepository

```
package sit.int202.simple.repositories;
:
public class SubjectRepository {
    private static List<Subject> subjects;

    public List<Subject> findAll() {
        return subjects;
    }

    public SubjectRepository() {
        initialize();
    }

    private void initialize() {
        subjects = new ArrayList<>(20);
        :
        :
        :
        subjects.add(new Subject("INT 100", "IT Fundamentals", 3));
        subjects.add(new Subject("INT 207", "Network I", 3));
    }
}
```

SubjectListServlet

```
package sit.int202.simple.servlet;
:
@WebServlet(name = "SubjectListServlet", value = "/SubjectList")
public class SubjectListServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        List<Subject> subjects = (List) new SubjectRepository().findAll();
        request.setAttribute("subjects", subjects);

        request.getRequestDispatcher("/subject_listing.jsp").forward(request, response);
    }

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

View: subject_listing.jsp (1)

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>Subject Listing</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/css/bootstrap.min.c
ss" rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/js/bootstrap.bundle.
min.js"></script>
</head>
</head>
<body style="margin-left: 20px">
<div class="container-fluid">
    <h3>Subject Listing:</h3>
    <div class="row bg-light">
        <div class="col-1">No.</div>
        <div class="col-1">Subject Id</div>
        <div class="col-3">Subject Title</div>
        <div class="col-1">Credit</div>
        <div class="col-6">Note</div>
    </div>
```

View: subject_listing.jsp (2)

```
<c:forEach items="${subjects}" var="subject" varStatus="vs">
  <div class="row">
    <div class="col-1">${vs.count}</div>
    <div class="col-1">${subject.id}</div>
    <div class="col-3">${subject.title}</div>
    <div class="col-1">${subject.credit}</div>
    <div class="col-6"></div>
  </div>
</c:forEach>
</div>
</body>
</html>
```

Output

← → ↻ localhost:8080/simple_war_exploded/SubjectList

Subject Listing:

No.	Subject Id	Subject Title	Credit	Note
1	INT 100	IT Fundamentals	3.0	
2	INT 101	Programming Fundamentals	3.0	
3	INT 102	Web Technology	1.0	
4	INT 114	Discrete Mathematics	3.0	
5	GEN 101	Physical Education	1.0	
6	GEN 111	Man and Ethics of Living	3.0	
7	LNG 120	General English	3.0	
8	LNG 220	Academic English	3.0	
9	INT 103	Advanced Programming	3.0	
10	INT 104	User Experience Design	3.0	
11	INT 105	Basic SQL	1.0	
12	INT 107	Computing Platforms Technology	3.0	
13	INT 200	Data Structures and Algorithms	1.0	
14	INT 201	Client-Side Programming I	2.0	
15	INT 202	Server-Side Programming I	2.0	
16	INT 205	Database Management System	3.0	
17	INT 207	Network I	3.0	