UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SECJ 3303 – INTERNET PROGRAMMING

TOPIC 02 – Integrating Servlets, JSP, and Java POJO: The Model View Controller (MVC)

innovative • entrepreneurial • global

UTM JOHOR BAHRU

OBJECTIVES

TOPIC 02

Knowledge

- Understand the Model-View-Controller (MVC) design pattern and how it is applied in Java web applications.
- Explain the roles of Servlets, JSP, and POJOs in an MVC-based application.
- Describe how data flows between the Model, View, and Controller layers in a Java web application.
- Understand how routing and URL mapping work in Java web applications using Servlets.

OBJECTIVES

TOPIC 02

Applied

- Develop a simple web application using the MVC pattern, integrating Servlets as controllers, JSP as the view, and Java POJOs as the model.
- Implement the flow of data between the controller (Servlet) and the view (JSP) through request attributes.
- Create Servlets that handle HTTP GET and POST requests for form submissions and processing.
- Use JSP to display dynamic data, and integrate routing for different web pages within the application.

TOPIC 02

PART 1 : MVC

TOPIC 02

Model View Controller (MVC) Design Pattern

Using MVC design pattern, an application is separated into three main logical components. MVC improves application organization by separating concerns.

1. **Model –** Manages the data and business logic.
2. **View –** Responsible for displaying the data to the user (UI).
3. **Controller –** Handles user input, processes the business logic using the model, and updates the view accordingly.

MVC pattern helps in creating scalable and maintainable web applications.

Components Breakdown:

Model:

- Represents data and the business rules or logic of the application.
- Acts as a gateway to the database, performing data operations.
- Example: Java POJOs (Plain Old Java Objects) or Java Beans.

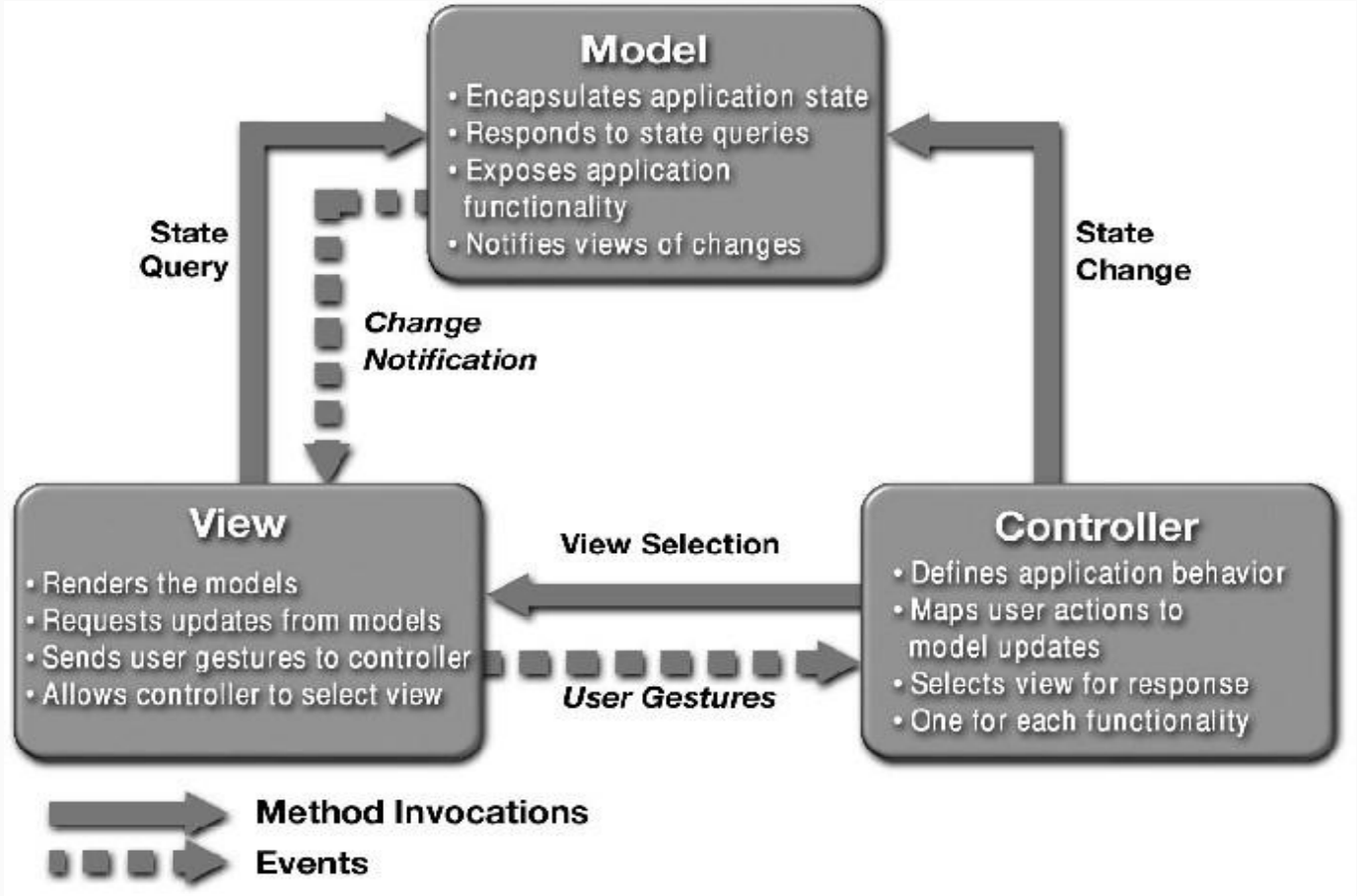
View:

- Responsible for rendering the user interface.
- Displays data provided by the Model in a user-friendly format.
- Example: JSP (JavaServer Pages) or Thymeleaf in Java web applications.

Controller:

- Manages user inputs (such as form submissions).
- Processes the input by calling the Model to perform business operations.
- Determines the appropriate view to render in response.
- Example: Java Servlets

MVC Pattern


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Example: User Registration Workflow

This workflow illustrates the process of registering a user in a web application using JSP, Servlet, and Model (POJO). It follows the MVC (Model-View-Controller) pattern, where the user interacts with the View, the Controller manages the request, and the Model handles business logic.

Step 1:

- View (JSP): The user fills in a registration form (e.g., username, email, password) on a JSP page.
- When the user submits the form, the data is sent as an HTTP request to the Controller (Servlet) ie via a POST request.

Step 2:

- Controller (Servlet): The Servlet receives the form data and processes it (e.g., validating inputs, checking if the username already exists).
- The Servlet interacts with the Model to handle business logic, such as saving the user's data.

Example (cont...)

Step 3:

- Model (POJO): The Model performs the necessary business operations (e.g., saving the user's details to the database or calculating any business rules).
- After processing, the Model returns the results to the Controller.

Step 4:

- Controller (Servlet): The Controller processes the outcome from the Model and decides which View (JSP) to display (e.g., success or error page).
- The Servlet uses RequestDispatcher to forward the request to the appropriate JSP page.

Step 5:

- View (JSP): The JSP displays the result to the user, either showing a success message or an error (e.g., “Registration Successful” or “Username already exists”).

Another Example - MVC Impl for User Login Use Case

TOPIC 02

Components:

LoginController.java : This is controller part of the application which communicates with model

Auth.java : The component which contains the business logic for authentication

User.java : The model which stores username and password for the user

errorPage.jsp : The view part- If login is not successful then this page is displayed

loginPage.jsp : The page with input form - username and password

adminPage.jsp : If login as admin is successful, then this page is displayed
coachPage.jsp : If login as coach is successful then this page is displayed

traineePage.jsp :- If login as registered trainee is successful, then this page is displayed

Steps:

Create a new Dynamic web project in STS by clicking

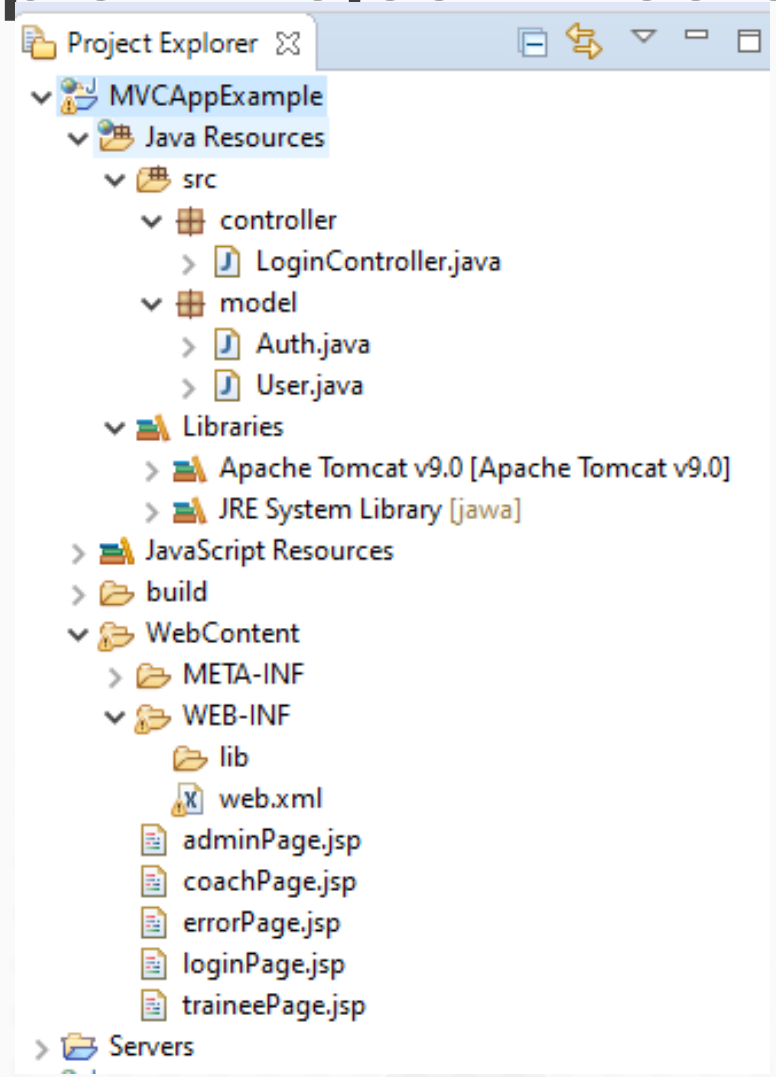
File > New > Dynamic Web Project.

Fill the details i.e. project name, the server. Enter your project name as "MVCAppExample". You will get the following directory structure for the project

Create LoginController.java (controller), Auth.java (model), User.java (model) loginPage.jsp, errorPage.jsp, adminPage.jsp,coachPage.jsp and traineePage.jsp (view) in the corresponding packages

Now that we have file structure, put the following code in corresponding files (to be completed: in-class activity)

MVCAppExample – Project Directory Structure



TOPIC 02

PART 2 : Servlet (as controller)

TOPIC 02

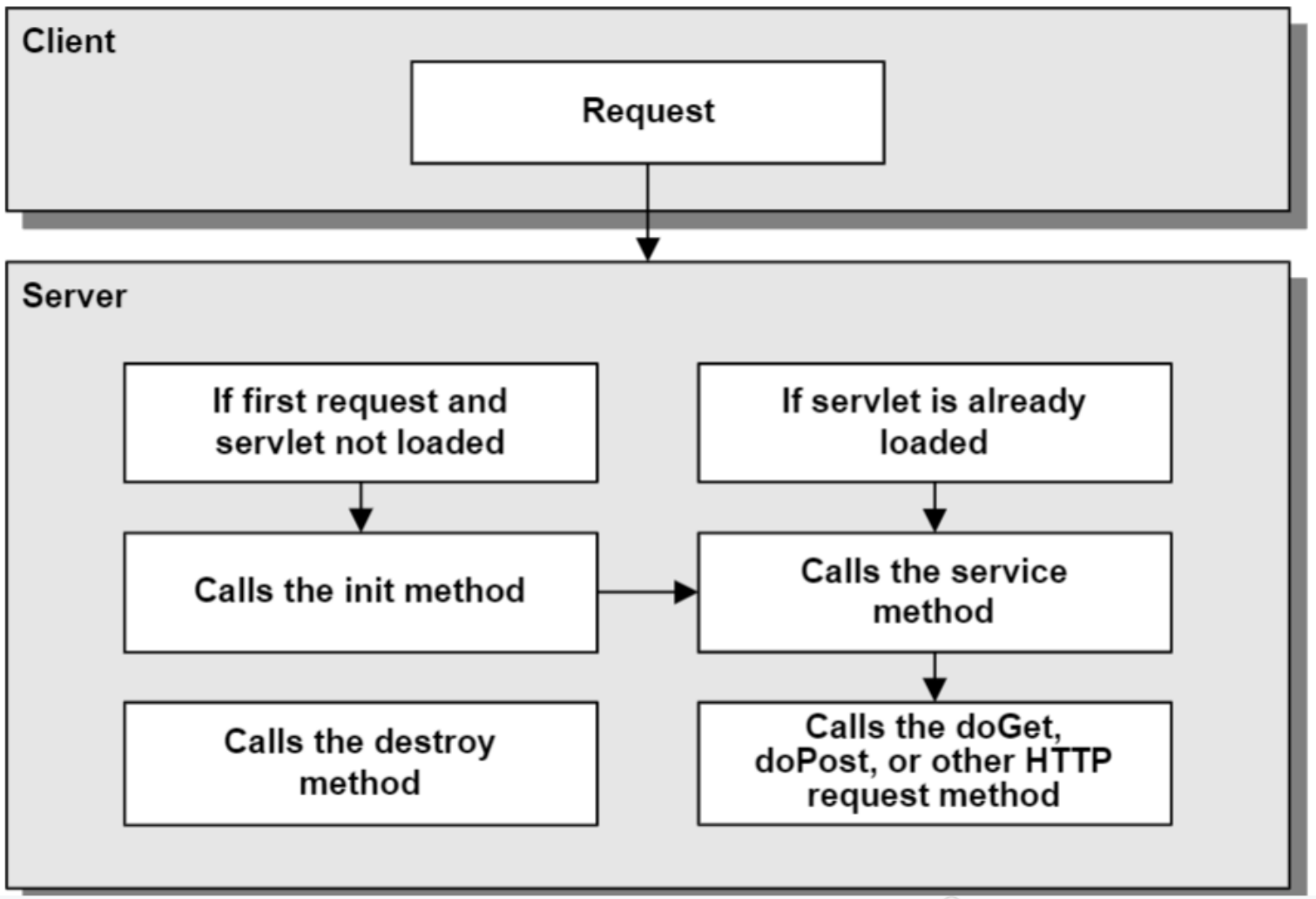
How Servlets Work in the MVC Pattern

Servlet - is a Java class that handles HTTP requests and responses in a web application
In MVC pattern, Servlet acts as the Controller, receiving input from the user (via forms or URL), processing the request, and determining the next step (forwarding data to the View or Model)

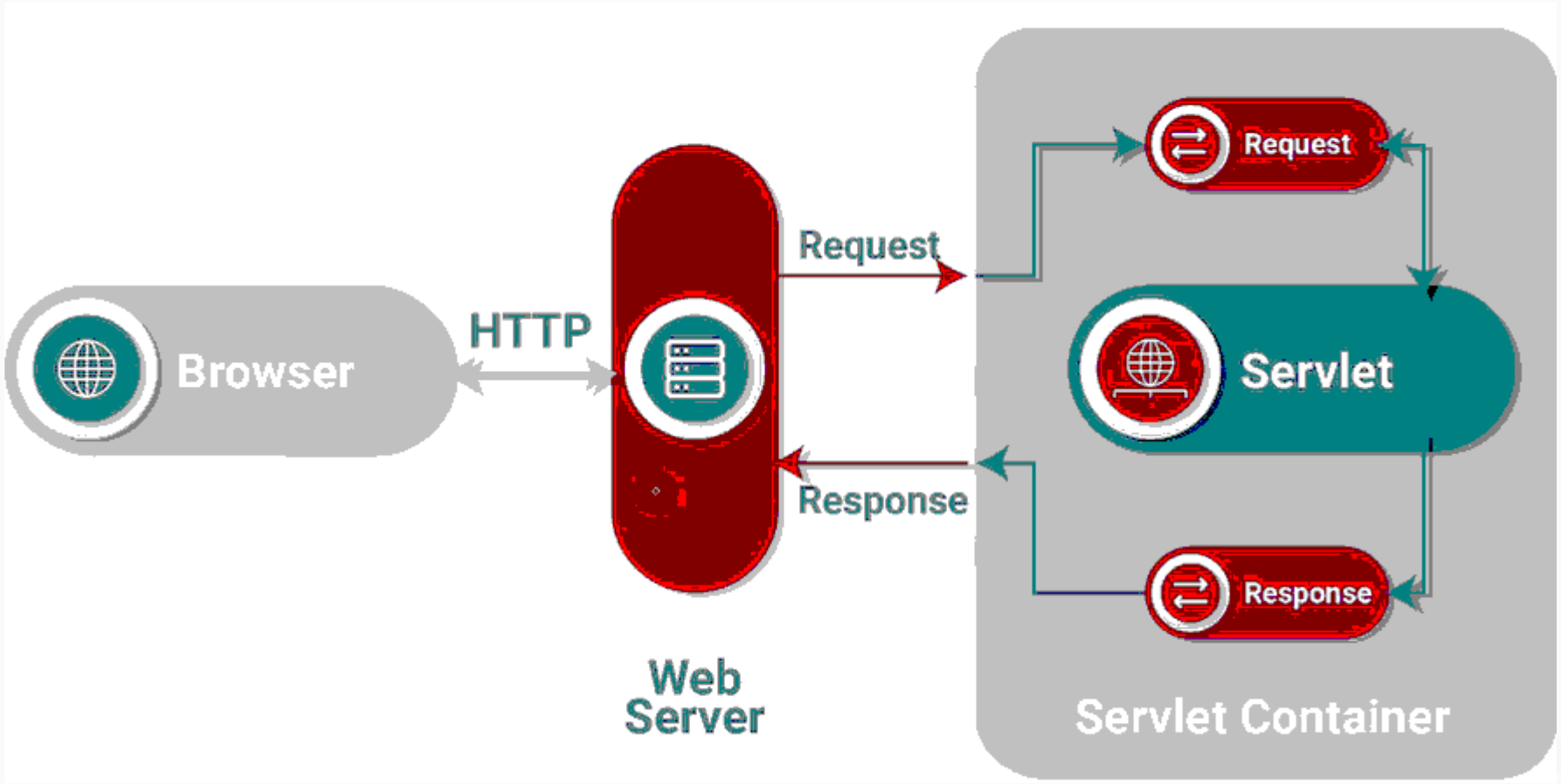
Servlet Lifecycle:

1. init(): Initializes the Servlet when it is first created.
2. service(): Handles requests and routes them to doGet() or doPost().
3. doGet(): Processes HTTP GET requests (e.g., retrieving data).
4. doPost(): Processes HTTP POST requests (e.g., form submissions).
5. destroy(): Cleans up resources when the Servlet is destroyed.

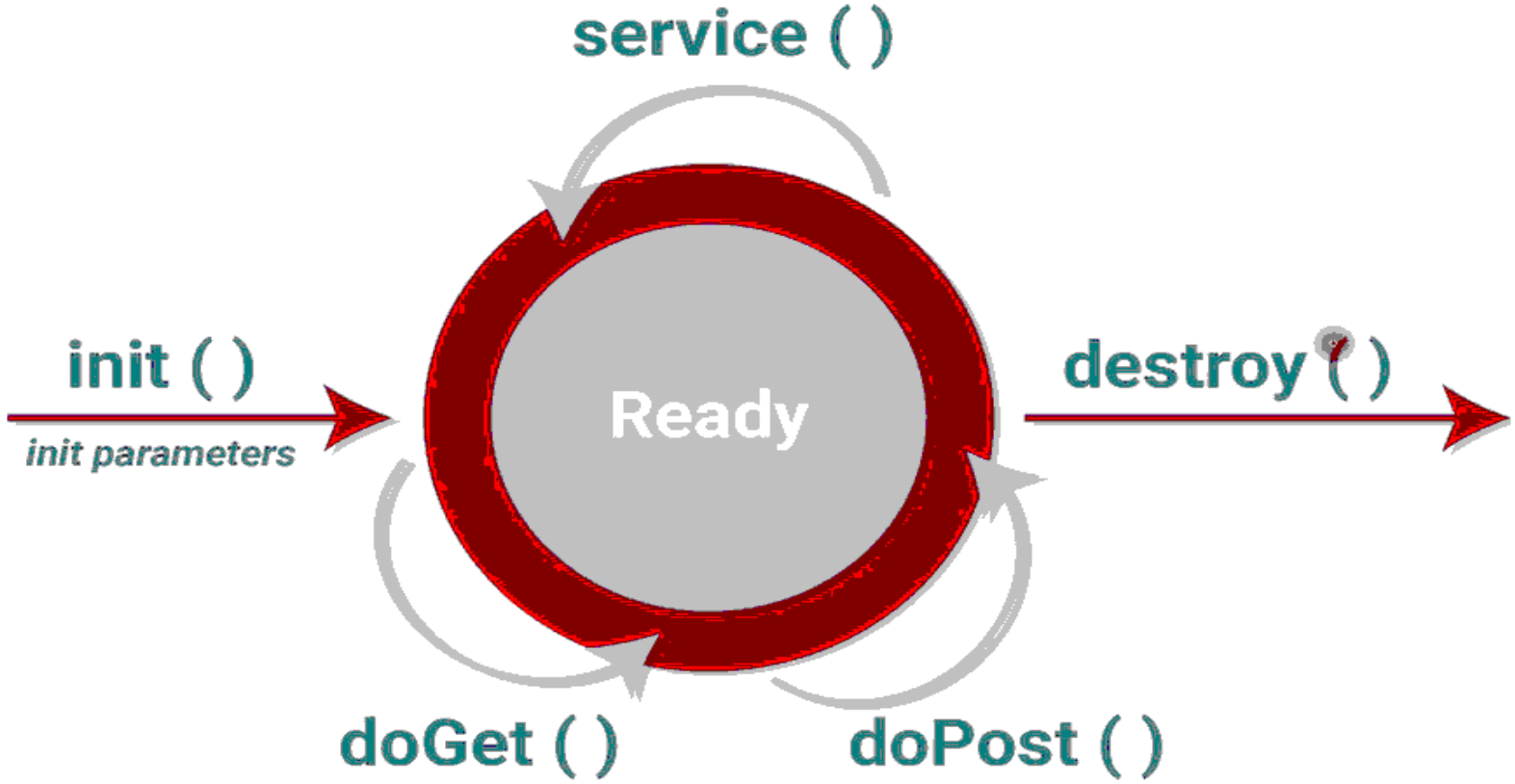
The lifecycle of a servlet



The lifecycle of a servlet



The lifecycle of a servlet



Note

- It's generally considered a bad practice to override the service method. Instead, we should override a method like doGet or doPost to handle a specific type of HTTP request.

The *doGet()* method

- is used to handle HTTP GET requests
- commonly used to retrieve information from the server
- Data is typically passed through the URL as query parameters in GET requests.

Use *doGet()* when :

- You need to retrieve data (e.g., display a form or a list of items)
- You want to request resources like HTML pages, images, etc...
- Query parameters are passed in the URL (e.g., PersonBMI?weight=77&height=1.77).

Code Example


```
public class HelloServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      // Read username parameter from the request
      String username = request.getParameter("username");

      //prepare the response/output
      // Set content type to HTML
      response.setContentType("text/html");

      / // Write HTML response with a personalized greeting
      response.getWriter().println("<h1>Hello, " + username + "! Welcome to the Servlet
Example!</h1>");    }
}
```

Note:

HttpServletRequest: Provides access to request data, including query parameters.

HttpServletResponse: Sends response data back to the client, including content like HTML, JSON, or text.

The *doPost()* method

Forwarding Requests from Servlet to JSP (from Controller to View)

TOPIC 02

1. Why Forward Requests to JSP?

- In MVC, the Servlet (Controller) processes the user input and business logic but forwards the final output to the JSP (View) to display the result to the user.
- This keeps the presentation logic separate from the business logic.

2. Using **RequestDispatcher** for Forwarding

- The RequestDispatcher object is used to forward the request from the Servlet to another resource, usually a JSP page.
- Forwarding passes both the request and response objects to the target resource without modifying the URL

RequestDispatcher Example:

```
// Set attributes to pass data to JSP
request.setAttribute("username", "safilani");
request.setAttribute("role", "admin");

// Forward the request to the JSP page
RequestDispatcher dispatcher = request.getRequestDispatcher("welcome.jsp");
dispatcher.forward(request, response);

//or shorter code
request.getRequestDispatcher("welcome.jsp").forward(request, response);
```

Note:

- **setAttribute()** method attaches data (username & role) to the request obj, which can then be accessed in the JSP.
- The **getRequestDispatcher()** method specifies the JSP page to forward to,
- **forward()** performs the actual redirection to the JSP.

TOPIC 02

PART 3 : POJO (as model)

POJO – Plain Old Java Object.

TOPIC 02

What is POJO?

- A simple Java object that doesn't follow any special conventions or frameworks.
- POJOs are typically used to represent data in a straightforward and easy-to-read manner
- Essential for representing data in Java applications, especially in the Model layer of MVC.
- POJO allow for clean separation between the business logic and the rest of the application

Characteristics of a POJO:

- No special inheritance: POJOs do not extend or implement any specific framework classes or interfaces.
- Encapsulation: POJOs follow the principle of encapsulation by using private fields and public getter/setter methods.
- No Annotations: POJOs do not require any annotations or other special configurations.

cont...

Why Use POJOs in MVC?

- Data Representation: POJOs are used in the Model layer of MVC to represent and encapsulate business data.
- Business Logic: POJOs can contain business logic, such as calculations or transformations.
- Easily Testable: POJOs are simple to test because they are not tied to any specific framework or external dependencies.

Advantages of POJOs:

- Simplicity: POJOs are easy to write and understand.
- Flexibility: Since POJOs are not tied to a specific framework, they can be used in any type of Java application.
- Reusability: POJOs can be reused across different layers or components of an application without modification.

Creating POJO

Key Components of a POJO:

- **Private Fields:** POJOs typically contain private variables that represent the object's data.
- **Public Getter and Setter Methods:** Methods that allow external access to private fields.
- **Constructors:** Used to create instances of the POJO, initializing the object with values.

cont...


```
public class User {
    private String username;       //private fields
    private String email;          //private fields

    // Constructor
    public User(String username, String email) {
        this.username = username;
        this.email = email;
    }

    // Getter methods
    public String getUsername() {
        return username;
    }

    public String getEmail() {
        return email;
    }

    // Setter methods
    public void setUsername(String username) {
        this.username = username;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

Another Example (Person BMI App)


```
public class BMICalculator {
    private double weight;  // Weight in kilograms
    private double height;  // Height in meters
    private double bmi;     // BMI value
    private String bmiCategory;

    // Constructor
    public BMICalculator(double weight, double height) {
        this.weight = weight;
        this.height = height;
        calculateBMI(); // Calculate BMI when the object is created
        determineBMICategory(); // Determine category
    }

    // Getters
    public double getWeight() {
        return weight;
    }

    public double getHeight() {
        return height;
    }

    public double getBMI() {
        return bmi; // Getter for BMI
    }

    public String getBMICategory() {
        return bmiCategory; // Getter for BMI category
    }

    // Business logic method to calculate BMI
    private void calculateBmi() {
        this.bmi = weight / (height * height);  // BMI calculation
    }

    //determine the BMI category based on the BMI value
    private void determineBMICategory() {
        if (bmi < 18.5) {
            this.bmiCategory = "Underweight";
        } else if (bmi >= 18.5 && bmi < 24.9) {
            this.bmiCategory = "Normal weight";
        } else if (bmi >= 25 && bmi < 29.9) {
            this.bmiCategory = "Overweight";
        } else {
            this.bmiCategory = "Obese";
        }
    }
}
```

Usage of POJO in a Web Application

TOPIC 02

- The User and BMICalculator POJO can be used in a Servlet to process user input and calculate the BMI.
- The results are then forwarded to a JSP for display.
- The User and BMI Calculator POJO demonstrates how to encapsulate business logic, such as calculations and classification, within a simple Java class.
- This separation of concerns ensures that the business logic is kept clean and reusable across different parts of the application

TOPIC 02

PART 4 : JSP (as view)

JSP (JavaServer Pages)

TOPIC 02

- Is a server-side technology that allows developers to create dynamic web pages.
- JSP combines HTML with embedded Java code to produce dynamic content.
- Typically used as the View layer in the MVC pattern to render data received from the Controller (Servlet).

How JSP Works:

- JSP files are compiled into Servlets by the server when they are first requested.
- Java code embedded in JSP (ie scriptlet, expression etc...) is executed on the server, resulting HTML is then sent to the client .mplementation of scriptlet, expression, declaration are not covered- will use EL & JSTL in the next topic.
- This allows JSP to dynamically generate content, such as user-specific data or responses based on server-side logic.

* Detail on Expression Language (EL) & JSP Standard Tag Library (JSTL) in next topic

Key Features of JSP:

- Dynamic content generation: JSP can include Java code (we will use EL & JSTL instead) to generate dynamic content.
- JSP provides EL and JSTL to access Java objects without writing Java code in the JSP.

How JSP Fits into the MVC Pattern:

- In the MVC architecture, JSP serves as the View layer:
- Servlet (Controller) processes the request and prepares the data.
- JSP (View) presents the data to the user in a readable format (usually HTML).
- This separation of concerns helps keep the application clean and maintainable.

Advantages of JSP:

- Integration of Java and HTML: JSP allows seamless integration of Java logic with HTML content.
- Simplifies server-side rendering: JSP makes it easy to dynamically render data on web pages.
- Reduces overhead: JSP pages are compiled into Servlets by the server, optimizing performance.

JSP Lifecycle:

JSP lifecycle involves several key phases that convert a JSP page into a Servlet and handle requests to render dynamic content.

The main stages of the lifecycle are:

- Translation: JSP is translated into a Servlet class.
- Compilation: The translated Servlet is compiled into bytecode.
- Execution: The compiled Servlet handles HTTP requests and generates dynamic content.

Example of Lifecycle in Action:

A simple example of a JSP being processed:

1. The user requests a JSP page (e.g., /welcome.jsp).
2. The JSP is translated into a Servlet (if not already).
3. The Servlet processes the request and generates dynamic HTML.
4. The resulting HTML is sent back to the user.

Accessing and Displaying Data in JSP:

- In the JSP, you can access the request attributes using Expression Language & JSTL, or Java code (ie scriptlet, expression, declaration...).

- Expression Language (EL) & JSTL are preferred for a cleaner and more readable approach:

Why Use Expression Language (EL) & JSTL?

- Cleaner syntax: EL JSTL makes it easier to access and display data without needing to write Java code directly in the JSP.
- Readability: EL JSTL expressions are simple and concise, making the JSP easier to read and maintain.
- Best practice: Using EL JSTL helps maintain the separation between presentation logic and business logic.

Code Example

```
<html>
<head>
   <title>User Information</title>
</head>
<body>
   <h1>Welcome, ${username}!</h1>
   <p>Thank you for logging in. Here is your personalized information:</p>

   <ul>
      <li>Username: ${username}</li>
      <li>Email: ${email}</li> //
      <!-- can add more here -->
   </ul>
</body>
</html>
```

Displaying Lists or Collections in JSP:

In addition to displaying individual attributes, JSP can also display lists or collections of data.

Example (using JSTL's <c:forEach> to loop over a collection):

```
<ul>
    <c:forEach var="product" items="${productList}">
        <li>${product.name}</li>
    </c:forEach>
</ul>
```

Lab Activity

Complete the implementation of Person BMI App & Car Loan Application

Notes: Try use the following
doGet() vs doPost()
request.getParameter() -vs- request.getParameterValues()
RequestDispatcher forward() -vs- response.sendRedirect()
request.setAttribute() -vs- session.setAttribute()

TOPIC 02

Java Web Application Frameworks

-detail in later topic {Spring Web MVC}

Web Application Frameworks

- Based on MVC Model 2 architecture
- Web-tier applications share common set of functionality
  - Dispatching HTTP requests
  - Invoking model methods
  - Selecting and assembling views
- Provide classes and interfaces that can be used/extended by developers

Web Application Frameworks

TOPIC 02

- Java Spring MVC (will be covered in next topic)

- Ruby on Rails
- PHP Laravel
- CodeIgniter, Etc...

Why Web Application Framework?

TOPIC 02

- De-coupling of presentation tier and business logic into separate components
- Provides a central point of control
- Provides rich set of features
- Facilitates unit-testing and maintenance
- Availability of compatible tools
- Provides stability
- Enjoys community-supports
- Simplifies internationalization Simplifies input validation

Why Web Application Framework?

TOPIC 02

-

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

TOPIC 02 – Integrating Servlets, JSP, and Java
POJO: The Model View Controller (MVC)

The End

innovative • entrepreneurial • global

UTM JOHOR BAHRU