UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SECJ 3303 – INTERNET PROGRAMMING

TOPIC 5 – JAVA WEB SESSION TRACKING

innovative • entrepreneurial • global

UTM JOHOR BAHRU

OBJECTIVES

Applied

- Provide for session tracking by using cookies, HTTP session and URL encoding.
- Provide for parameter passing by using URL rewriting and hidden fields.
- Test your web applications with cookies enabled and with cookies disabled.
- Write a utility class that includes a static method for getting a specific cookie from a user's browser.

Knowledge

- Describe the way HTTP works without session tracking.
- Describe the way cookies and HTTP session are used for session tracking.
- Describe the way URL encoding is used for session tracking.
- Distinguish between persistent cookies and per-session cookies.
- Distinguish between the use of URL rewriting and the use of hidden fields as ways to implement parameter passing.
- Describe the way HTTP session in servlet and JSP are used for session tracking.

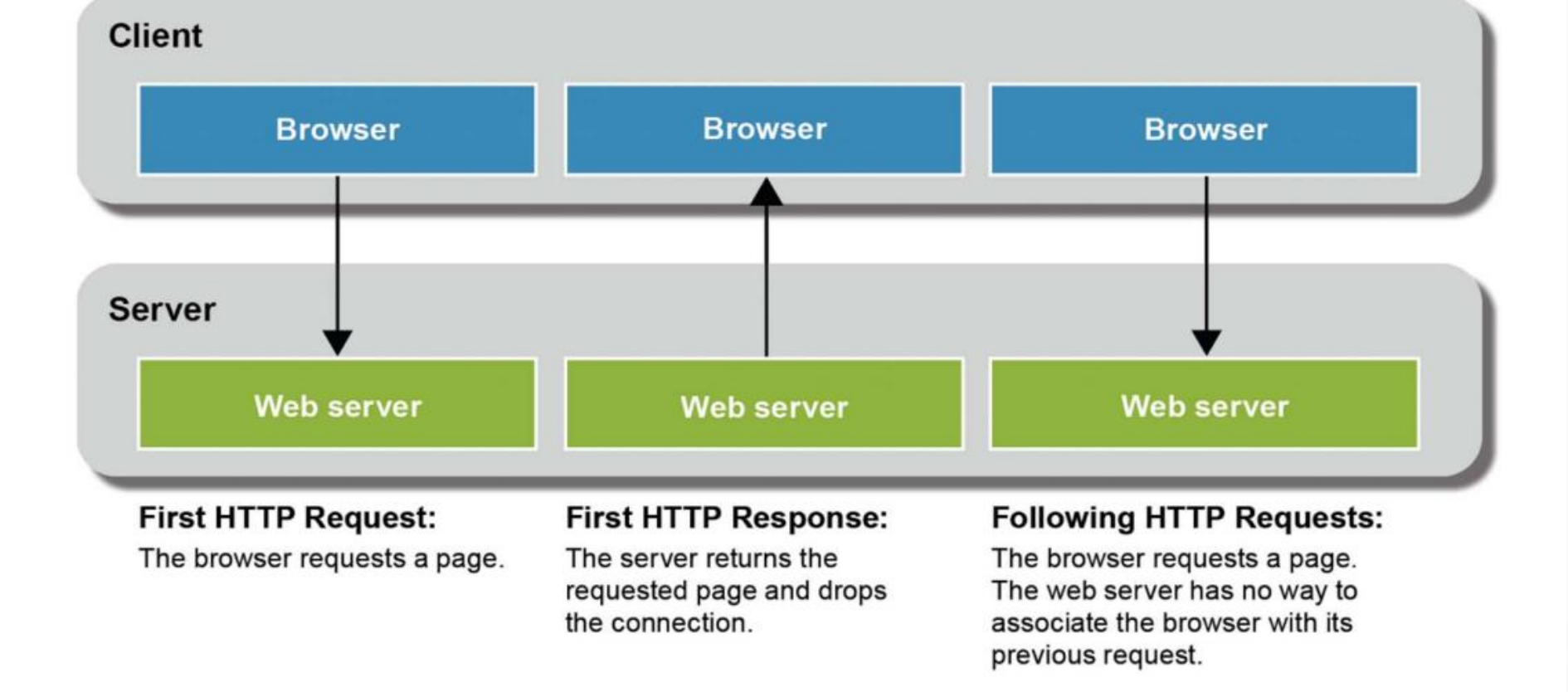
UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Session Tracking

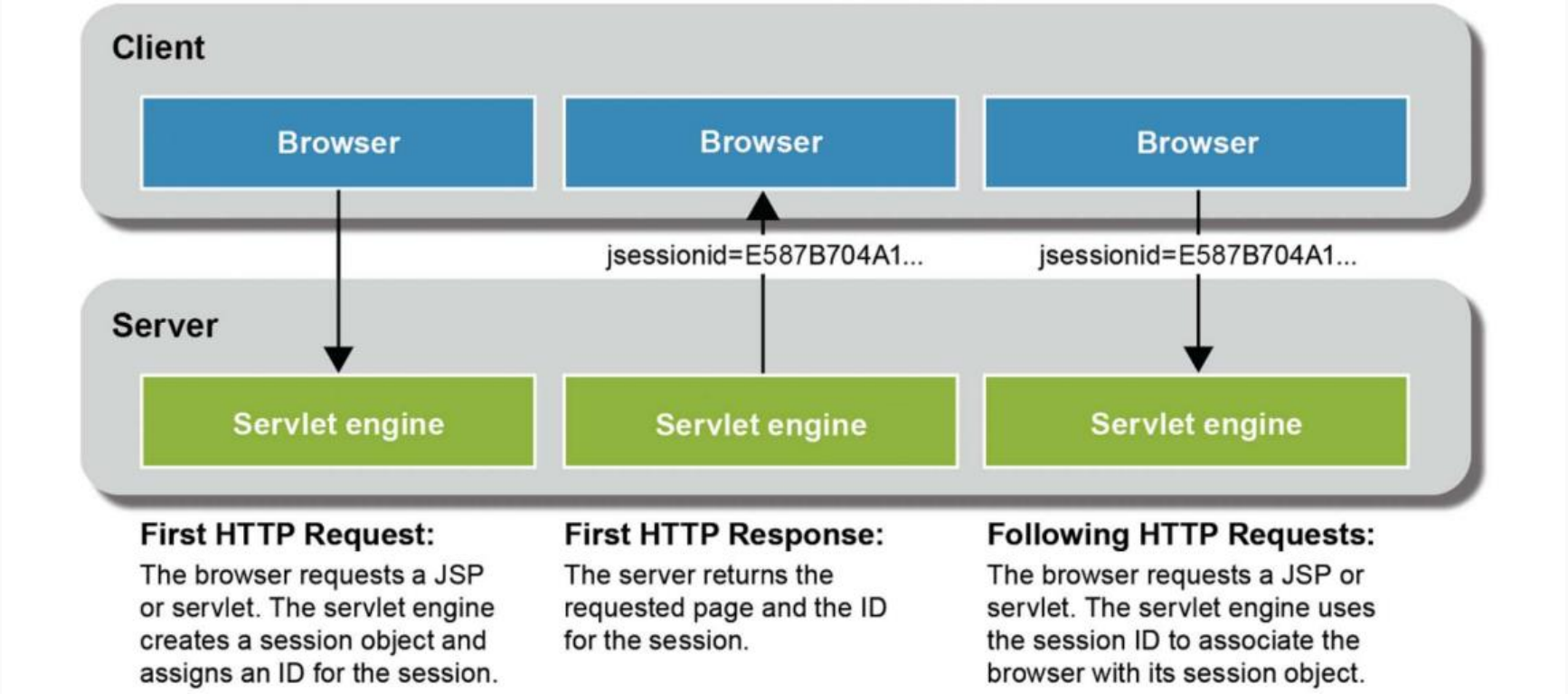
- **Session** is a particular interval of time.
- **Session Tracking** is a mechanism used by the **Web container** to store session information for a particular user.
- It is also known as a **Session Management** which is a way to maintain state (data) of the user.
- HTTP is a ***stateless protocol*** which means each request is considered as the new request. Once a browser makes a request, it drops the connection to the server. So, to **maintain *state***, a web application must use *session tracking*.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Why session tracking is difficult with HTTP?



How Java keep tracks of session?

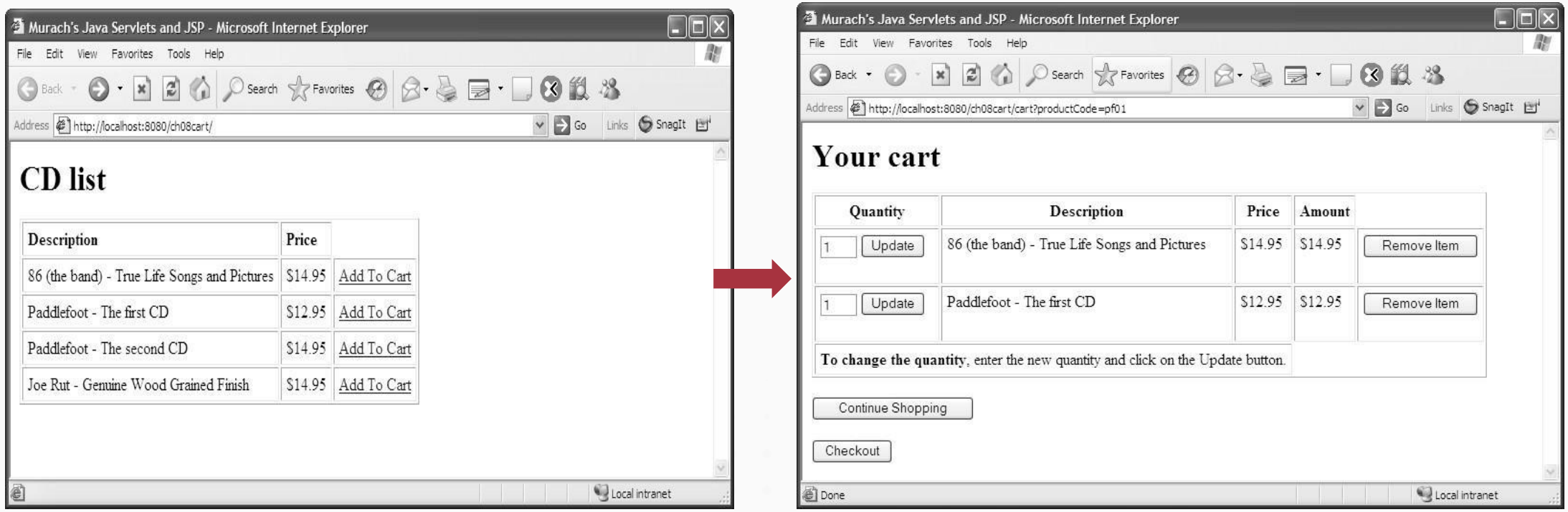


How Java keep tracks of session?

- A browser on a client request a JSP or servlet from the web server, which passes the request to the servlet engine.
- Then, the servlet engine checks if the request includes an ID for the Java session.
- If it doesn't, the servlet engine **creates a unique ID for the session plus a session objec**t that can be used to store the data for the session.
- From that point on, the web server uses the session ID to relate each browser request to the session object, even though the server still drops the HTTP connection after returning each page.

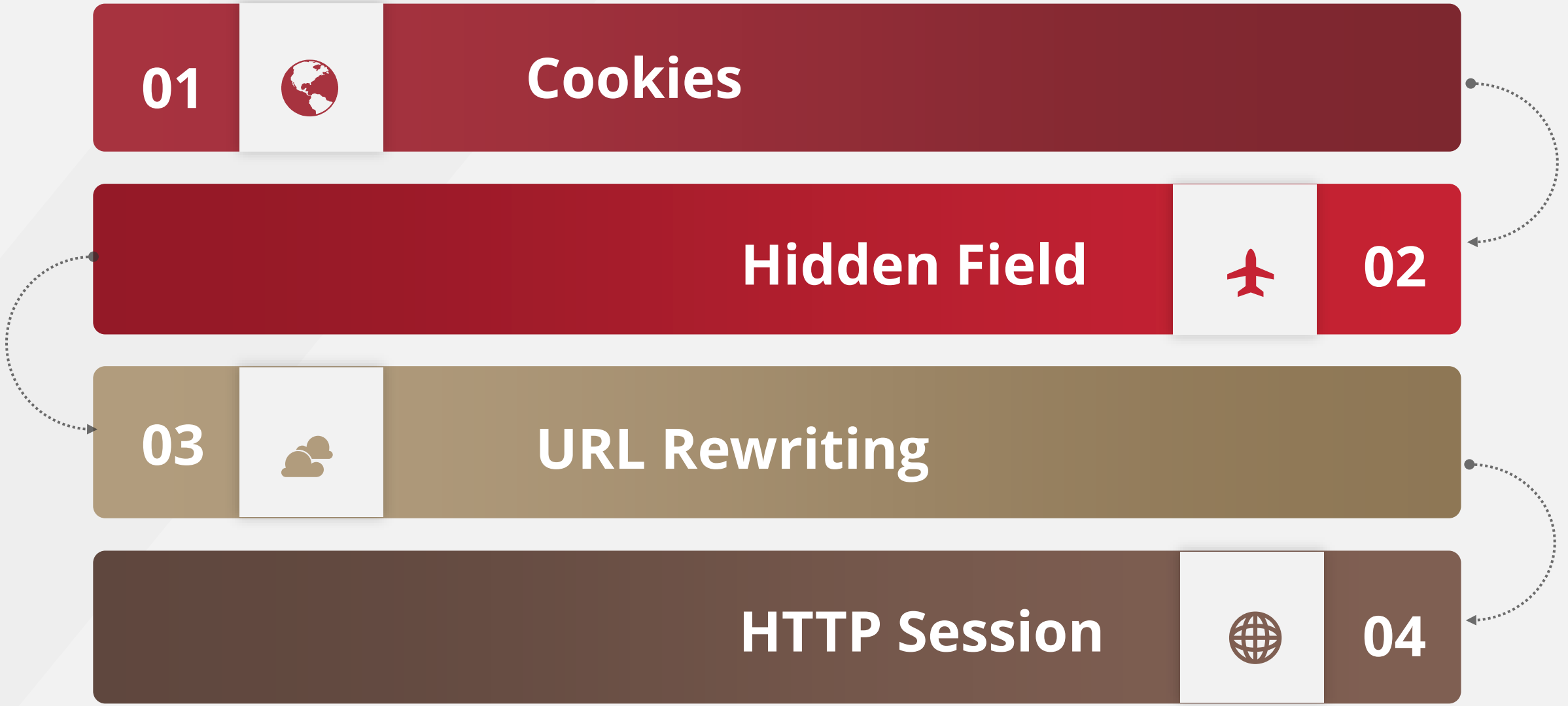
UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Example of session tracking



For example, a shopping cart module should know: who is sending the request to add an item, in which cart the item has to be added or who is sending checkout request. So that, it can charge the amount to the correct client.

Techniques for Session Tracking



Cookies

- A **cookie** is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.
- By default, the servlet API uses a *cookie* to store a session ID in each browser. Then, the browser passes the cookie to the server with each request. To store the data for each session, the server creates a *session object*.
- There are two types of cookies:
  - *persistent cookies* are store on the user's PC. It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or sign-out.
  - *per-session cookies (non-persistent)* are deleted when the session ends. It is valid for single session only. It is removed each time when user closes the browser.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

How cookies work?

- A cookie is a name/value pair that is stored in a browser.
- On the server, a web application creates a cookie and sends it to the browser.
- On the client, the browser saves the cookie and sends it back to the server every time it accesses a page from that server.
- Cookies can be set to persist within the user's browser for up to 3 years.
- Some users disable cookies in their browsers. As a result, you can't always count on all users having their cookies enabled.
- Browsers generally accept only 20 cookies from each site and 300 cookies total. In addition, they can limit each cookie to 4 kilobytes.
- A cookie can be associated with one or more subdomain names.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Typical uses for cookies

- **To allow users to skip login and registration forms** that gather data like user name, password, address, or credit card data.
- **To customize pages** that display information like weather reports, sports scores, and stock quotations.
- **To focus advertising** like banner ads that target the user's interests.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Cookies

Constructor of the Cookie class

Constructor	Description
`Cookie(String name, String value)`	Creates a cookie with the specified name and value.

A method of the response object

Method	Description
`addCookie(Cookie c)`	Adds the specified cookie to the response.

A method of the request object

`getCookies()`	Returns an array of Cookie objects that the client sent with this request. If no cookies were sent, this method returns a null value.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

The methods of the Cookie class

Cookies

Method	Description
setMaxAge(int maxAgeInSeconds)	To create a persistent cookie, set the cookie's maximum age to a positive number. To create a per- session cookie, set the cookie's maximum age to –1. Then, the cookie will be deleted when the user exits the browser.
setPath(String path)	To allow the entire application to access the cookie, set the cookie's path to "/".
getName()	Returns a string for the name of the cookie.
getValue()	Returns a string that contains the value of the cookie.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Code that creates and sets a cookie

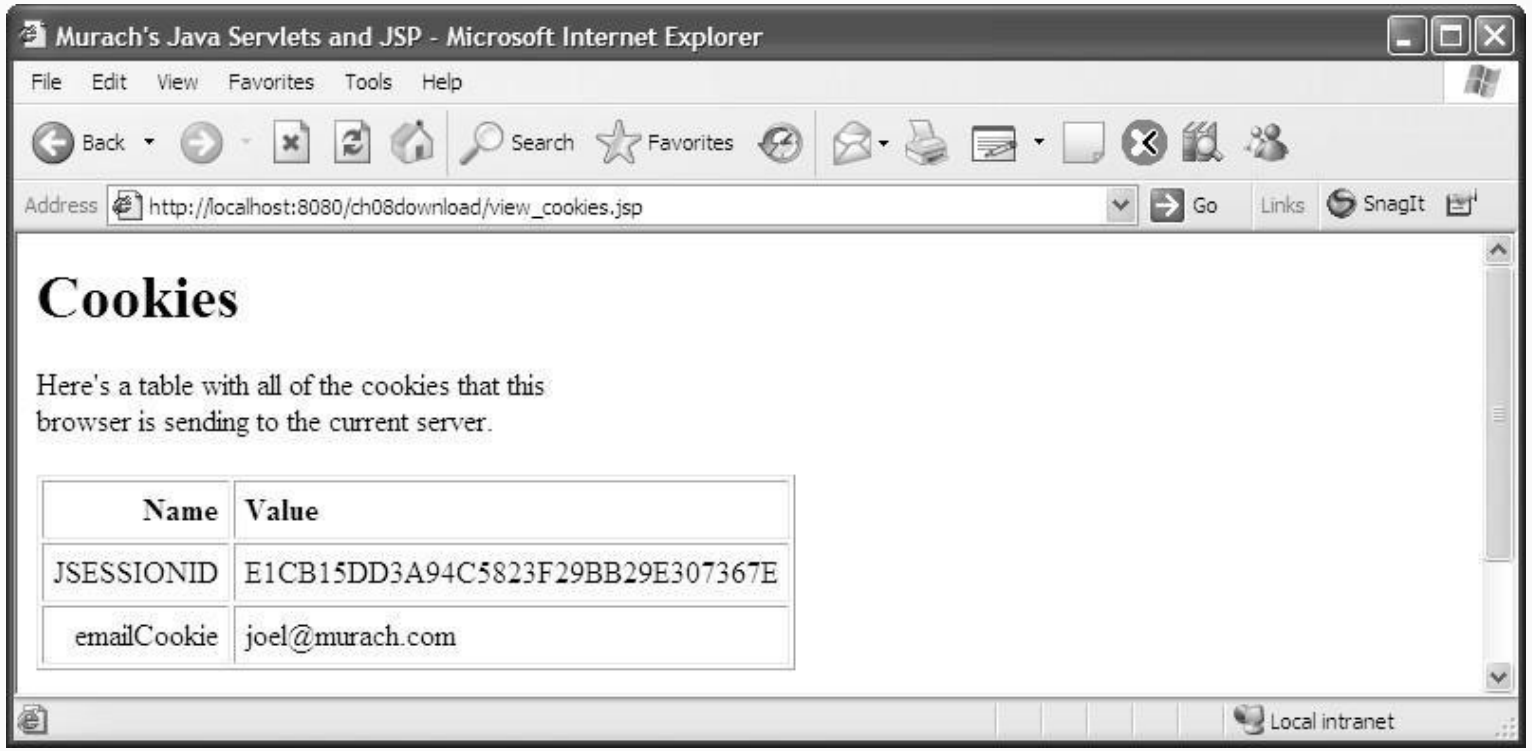
```
Cookie userIdCookie = new Cookie("userIdCookie", userId);
userIdCookie.setMaxAge(60*60*24*365*2);   //set the age to 2 years
userIdCookie.setPath("/");          // allow access by the entire application
response.addCookie(userIdCookie);
```

Code that gets the cookie

```
Cookie[] cookies = request.getCookies();
String cookieName = "userIdCookie";
String cookieValue = "";

for (int i=0; i<cookies.length; i++)
{
    Cookie cookie = cookies[i];
    if (cookieName.equals(cookie.getName()))
      cookieValue = cookie.getValue();
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

A JSP that shows all cookies for the current server



JSP code that displays all cookies

```
<%
  Cookie[] cookies = request.getCookies();
  for (Cookie c : cookies)
  {
%>
  <tr>
    <td align="right"><%= c.getName() %></td>
    <td><%= c.getValue() %></td>
  </tr>
<%
  }
%>
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Servlet code that deletes all persistent cookies


```
Cookie[] cookies = request.getCookies();
for (int i=0; i<cookies.length; i++)
{
    Cookie cookie = cookies[i];
    cookie.setMaxAge(0); //delete the cookie
    cookie.setPath("/");
            //allow the entire application to access it
    response.addCookie(cookie);
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Four methods of the Cookie class

- **`setPath(`**`String path`**`)`**
- **`setDomain(`**`String domainPattern`**`)`**
- **`setSecure(`**`boolean flag`**`)`**
- **`setVersion(`**`int version`**`)`**

Note

- All of these set methods have corresponding get methods.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

A utility class that gets the value of a cookie


```
package util;import
javax.servlet.http.*;

public class CookieUtil
{
    public static String
        getCookieValue(Cookie[] cookies,
        String cookieName)
    {
        String cookieValue = "";
        Cookie cookie;
        if (cookies != null)
        {
            for (int i=0; i<cookies.length; i++)
            {
                cookie = cookies[i];
                if (cookieName.equals(cookie.getName()))
                {
                    cookieValue = cookie.getValue();
                }
            }
        }
        return cookieValue;
    }
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Code that uses the CookieUtil class to get the value of a cookie

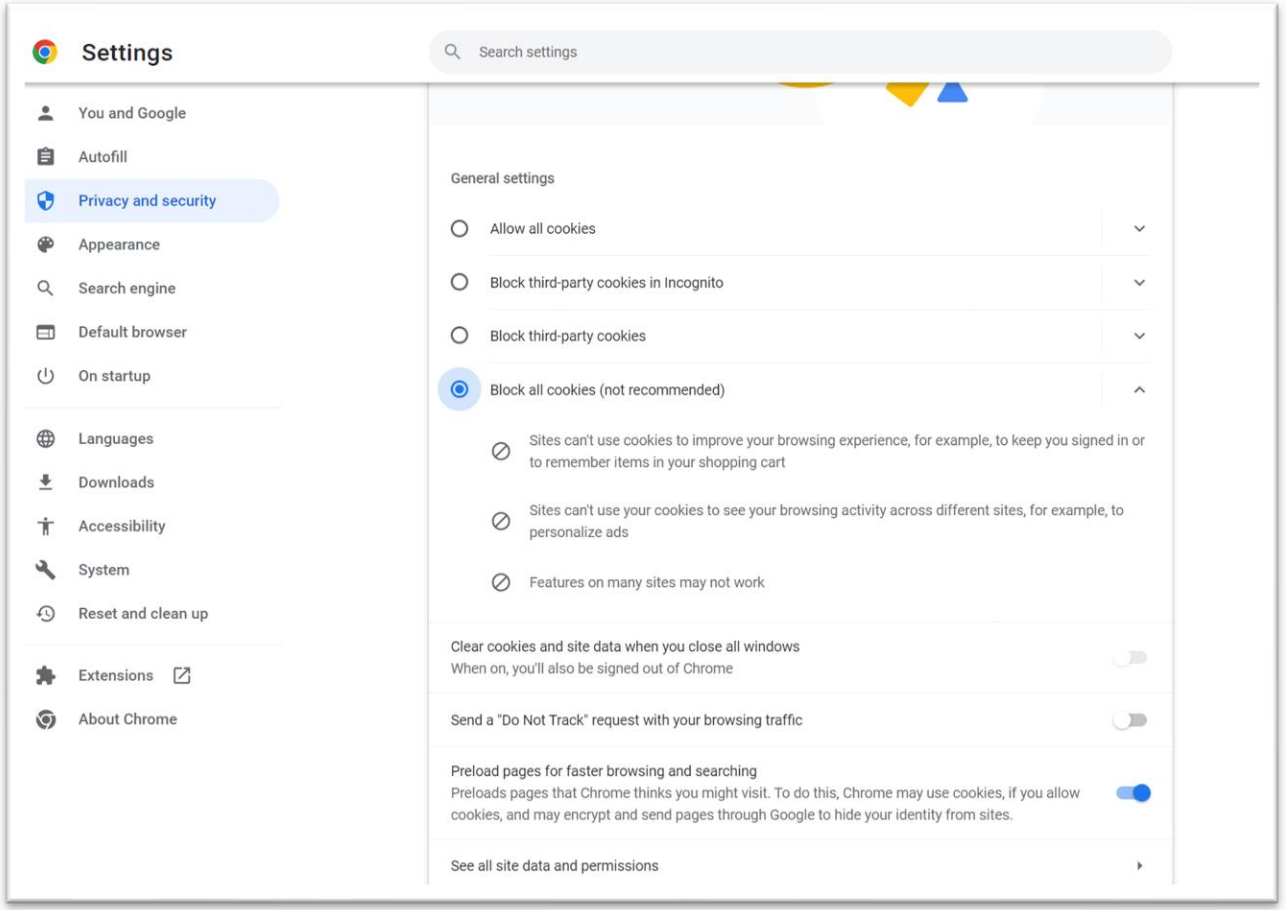
```
Cookie[] cookies = request.getCookies();
String emailAddress =
    CookieUtil.getCookieValue(cookies, "emailCookie");
```

To make it easier to get the value of a cookie, you can create a **utility class that contains a method that accepts an **array of Cookie object** and the **name of the cookie**, and then **returns the value of the cookie**.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

How to disable cookies in Google Chrome browser?

Cookies



1. On your computer, open Chrome.
2. At the top right, click More Settings.
3. Under "Privacy and security," click Site settings.
4. Then select - Cookies and other site data.
5. Select Block all cookies.

Your browser will no longer store cookies.

More info on how to disable cookies in other web browsers:
https://www.avg.com/en/signal/disable-cookies

Advantages and Disadvantages Cookies

Advantages of using cookies

- Cookies are simple to use and implement.
- Occupies less memory, do not require any server resources and are stored on the user's computer.
- We can configure cookies to expire when the browser session ends (session cookies) or they can exist for a specified length of time on the client's computer (persistent cookies).
- Cookies persist a much longer period of time than session state.

Disadvantages of using cookies

- User has the option of disabling cookies on his computer from browser's setting.
- Cookies will not work if the security level is set to high in the browser.
- Users can delete a cookies.
- User's browser can refuse cookies, so your code has to anticipate that possibility.
- Complex type of data is not allowed (e.g. dataset etc). It allows only plain text (cookie allows only string content).

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Hidden Form Field

Hidden Field

- A hidden field is used to store client state. In this case, user information is stored in hidden field value and retrieved from another servlet.
- The server embeds new hidden fields in every dynamically generated form page for the client.
- When the client submits the form to the server, the hidden fields identify the client.
- Web server can send a hidden HTML form field along with a unique session ID as follows:

```
<input type = "hidden" name = "sessionid" value = "12345">
```

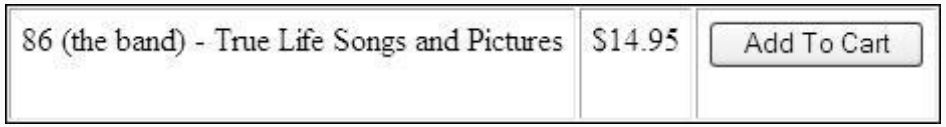

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Example of Hidden Form Field

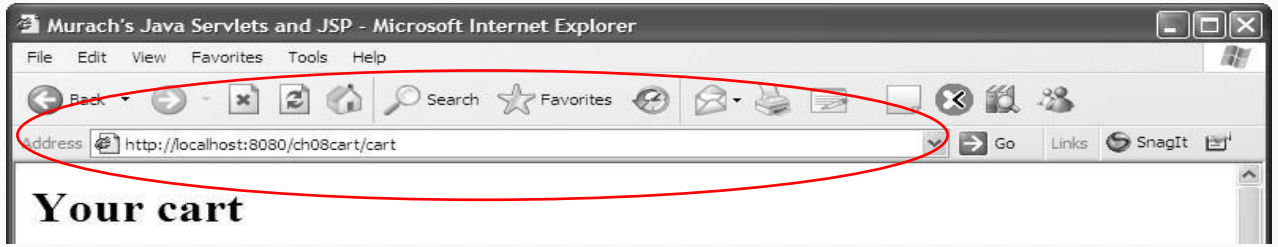
A Form tag that uses a hidden text field

```
<form action="cart" method="post">
  <input type="submit" value="Add To Cart">
  <input type="hidden" name="productCode" value="8601">
</form>
```

The form displayed in a browser



The URL that displays when the button is clicked



More Example of Hidden Form Field

A Form tag that uses JSP expressions to set hidden field values

```
<form action="cart" method="post">
    <input type="hidden" name="productCode"
       value="<%=product.getCode()%>">
    <input type=text size=2 name="quantity"
       value="<%=lineItem.getQuantity()%>">
    <input type="submit" name="updateButton" value="Update">
</form>
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Advantages and Disadvantages Hidden Form Field

Advantages of using hidden form field

- It will always work whether cookie is disabled or not.
- Hidden boxes reside in web pages of the browser windows, so they do not provide a burden to the server.

Disadvantages of using hidden form field

- It is maintained at server side.
- Extra form submission is required on each pages.
- Only textual information can be used.
- The hidden box values of the form page can be viewed using the source code of the web page. That means there is no security and are not appropriate for secure data like passwords.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

URL Rewriting

- URL rewriting is a method of session tracking in which some extra data (session ID) is appended at the end of each URL.
- This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session.
- This method is used with browsers that do not support cookies or where the user has disabled the cookies.
- In URL rewriting, a token(parameter) is added at the end of the URL as follow:

format : `url?name1=value1&name2=value2`
example : `hello?sessionid=12345&user=ali`

- A name and a value is separated using an equal (=) sign, a parameter name/value pair is separated from another parameter using the ampersand(&).

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Example of URL rewriting

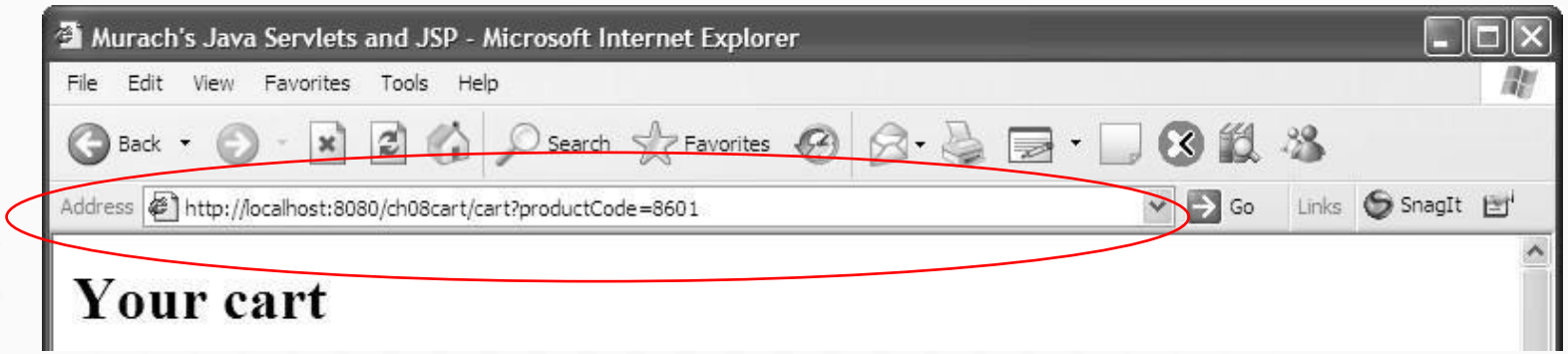
- **An A tag that adds a product code to a URL**

```
<a href="cart?productCode=8601">Add to cart</a>
```

- **The link displayed in a browser**

86 (the band) - True Life Songs and Pictures	$14.95	Add To Cart

- **The URL that displays when you click on the link**



More examples of URL Rewriting

A Form tag that calls a JSP

```
<form action="cart.jsp?productCode=jr01" method="post">
```

An A tag that uses a JSP expression for the product code

```
<a href="cart?productCode=<%= productCode %>" >
Add to cart</a>
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

How to use URL encoding to track sessions if cookies is disabled?

- If the user has disabled per-session cookies, you can use ***URL encoding*** to keep track of the ID for the session. To do that, you must convert any relevant HTML pages to JSPs, and you must encode all relevant URLs.

- When you encode a URL, the session ID is passed to the browser in the URL.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

A method to encode a URL

Method	Description
`encodeURL(String url)`	Returns a string for the specified URL. If necessary, this method encodes the session ID in the URL. If not, it returns the URL unchanged.

How to encode a URL in a Form tag

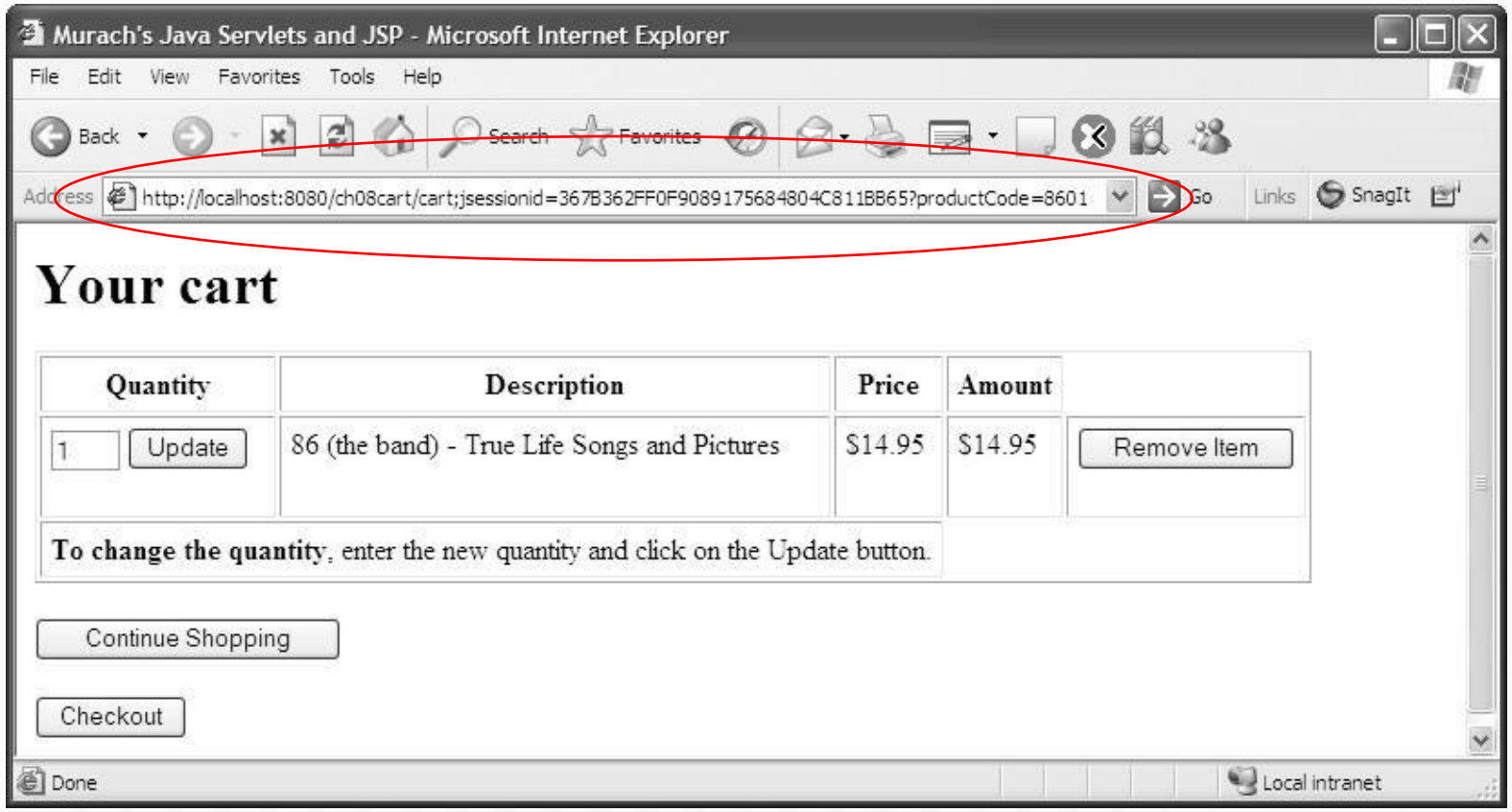
```
<form action="<%=response.encodeURL("cart")%>"
      method="post">
```

How to encode a URL in an A tag

```
<a href="<%=response.encodeURL("cart?productCode=8601")%>">
Add To Cart
</a>
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

A URL after it has been encoded



Advantages and Disadvantages URL Rewriting

Advantages of using URL Rewriting

- It will always work whether cookie is disabled or not.
- Extra form submission is not required on each pages.

Disadvantages of using URL Rewriting

- It will work only with links.
- It can send only textual information.
- Most browsers limit the number of characters that can be passed by a URL to 2,000 characters.
- It's difficult to include spaces and special characters such as the ? and & characters in parameter values.
- Every time we need to rewrite the URL with session-id value in the generated form, for this we must execute the encoded URL() method.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

HTTP Session - Servlet

- `HttpSession` object is used to store entire session with a specific client. We can store, retrieve and remove attribute from `HttpSession` object.
- Any servlet can have access to `HttpSession` object throughout the `getSession()` method of the `HttpServletRequest` object.
- The `HttpServletRequest` interface provides two methods to get the object of `HttpSession`:
  - **`public HttpSession getSession()`**: Returns the current session associated with this request, or if the request does not have a session, creates one.
  - **`public HttpSession getSession(boolean create)`**: Returns the current `HttpSession` associated with this request or, if there is no current session and create is true, returns a new session.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

HTTP Session - JSP

- By default, JSPs have session tracking enabled and a new `HttpSession` object is instantiated for each new client automatically.
- Disabling session tracking requires explicitly **turning it off by setting the page directive session attribute to false** as follows:

```
<%@ page session = "false" %>
```

- The JSP engine exposes the `HttpSession` object to the JSP programmer through the implicit **session** object.
- Since **session** object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or **`getSession()`.**

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Method for request object

Method	Description
`getSession()`	• Returns the HttpSession object associated with this request. If the request is not associated with a session, this method creates a new HttpSession object and returns it.

Methods for session object

Method	Description
`setAttribute(String name, Object o)`	Stores any object in the session as an attribute and specifies a name for the attribute.
`getAttribute(String name)`	Returns the value of the specified attribute as an Object type. If no attribute exists for the specified name, this method returns a null value.
`removeAttribute(String name)`	Removes the specified attribute from this session.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

How to set and get session attributes?

- A session object is created when a browser makes the first request to a site. It is destroyed when the session ends.
- A session ends when a specified amount of time elapses without another request or when the user exits the browser.
- The session object is a built-in JSP object. As a result, you don’t need to create the session object when working with JSPs.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Examples of code that...

Gets a session object

```
HttpSession session = request.getSession();
```

Sets a String object as an attribute

```
session.setAttribute("productCode", productCode);
```

Sets a user-defined object as an attribute

```
Cart cart = new Cart(productCode);
session.setAttribute("cart", cart);
```

Gets a String object

```
String productCode =
    (String) session.getAttribute("productCode");
```

Gets a user-defined object

```
Cart cart = (Cart) session.getAttribute("cart"); if
(cart == null)
    cart = new Cart();
```

Removes an object

```
session.removeAttribute("productCode");
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

More methods of the session object

Method	Description
`getAttributeNames()`	Returns a java.util.Enumeration object that contains the names of all attributes in the HttpSession object.
`getId()`	Returns a string for the unique Java session identifier that the servlet engine generates for each session.
`isNew()`	Returns a true value if the client does not yet know about the session or if the client chooses not to join the session.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

More methods of the session object (cont.)

Method	Description
`setMaxInactiveInterval``(int seconds)`	By default, the maximum inactive interval for the session is set to 1800 seconds (30 minutes). To increase or decrease this interval, supply a positive integer value. To create a session that won't end until the user closes the browser, supply a negative integer such as –1.
`invalidate()`	Invalidates the session and unbinds any objects that are bound to it.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Examples of code

A method that gets all the names of the attributes for a session

```
Enumeration names = session.getAttributeNames();
while(names.hasMoreElements())
{
    System.out.println((String) names.nextElement());
}
```

A method that gets the ID for a session

```
String jSessionId = session.getId();
```

A method that sets the inactive interval for a session

```
session.setMaxInactiveInterval(60*60*24); // one day
session.setMaxInactiveInterval(-1);  // until the browser is closed
```

A method that invalidates the session and unbinds any objects

```
session.invalidate();
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Examples of code

An example that synchronizes access to the session object

```
Cart cart;
synchronized(session)
{
    cart = (Cart) session.getAttribute("cart");
}
```

Another example that synchronizes access to the session object

```
synchronized(session)
{
    session.setAttribute("cart", cart);
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

A web browser with three windows accessing the same session object

How to provide thread-safe access to the session object?

- Each servlet creates one session object that exists for multiple requests that come from a single client.
- If the client has one browser window open, access to the session object is thread-safe.
- If the client has multiple browser windows open, it's possible (though highly unlikely) that two threads from the same client will access the session object at the same time.
- As a result, the session object isn't completely thread-safe.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Advantages and Disadvantages HTTP Session

Advantages of using HTTP Session

- There are no restrictions on the size of the object, any kind of object can be stored in a session.
- The usage of the session is not dependent on the client's browser.
- It is secure and transparent.

Disadvantages of using HTTP Session

- Performance overhead in case of large volumes of data/user, because session data is stored in server memory.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Summary

Summary

- HTTP is a stateless protocol, so web application must provide for session tracking.
- Session tracking enable an application to relate each request to a specific browser and to the data for that session.
- To provide for session tracking, Java creates one session object for each browser. Then, you can add attributes like variables and objects to this session object, and can retrieve the values of these attributes in any of the servlet and JSPs that are run during the session.
- There are four techniques session tracking – cookies, hidden field, URL rewriting and HTTP Session.
- In general, it is considered a best practice to implement session tracking by using cookies. The session ID is stored in a cookie on the user's browser. However, it doesn't work unless the browser enables cookies.
- It's also possible to implement session tracking by using URL encoding. This work even when the browser doesn't enable cookies.
- To pass parameters to a servlet, URL rewriting or hidden fields also can be used.
- HttpSession object is used to store entire session with a specific client.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

TOPIC 5 – Java Web Session Tracking

The End

Credit:
The content in this slide is based on textbook - Murach's Java Servlets/JSP (3rd Ed.)
© 2014, Mike Murach & Associates, Inc

innovative • entrepreneurial • global

UTM JOHOR BAHRU

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SECJ 3303 – INTERNET PROGRAMMING

TOPIC 7 – SPRING MVC

innovative • entrepreneurial • global

UTM JOHOR BAHRU

OBJECTIVES

Applied

- Test your web application using Apache Maven build tool.
- Write a code with Spring MVC framework.

Knowledge

- Describe the Apache Maven build tool.
- Describe the overview of Spring framework.
- Describe the Spring Web MVC in Spring framework.
- Understand the important components of Spring Web MVC.
- Explain the Spring Web MVC flow and know how to implement in a code.
- Understand the Spring Exception Handling

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Apache Maven

- Maven is a build automation and dependency management tool that helps in project management.
- **Build tool** is essential for the process of building. It is needed for the following process:
  - Generating source code
  - Generating documentation from the source code
  - Compiling of source code
  - Packaging the compiled code into JAR files.
  - Installing the package code in local repository, server or central repository.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Apache Maven

- Maven is written is Java and C# and is based on the **Project Object Model** (pom.xml).
- Project Object Model is an **XML file** that has all the information regarding project and configuration details.
- When we tend to execute a task, Maven searches for the POM in the current directory.
- The tool is used to build and manage any **Java-based project**.
- It helps in downloading dependencies, which refer to the libraries or JAR files.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

The Problem That Maven Solved

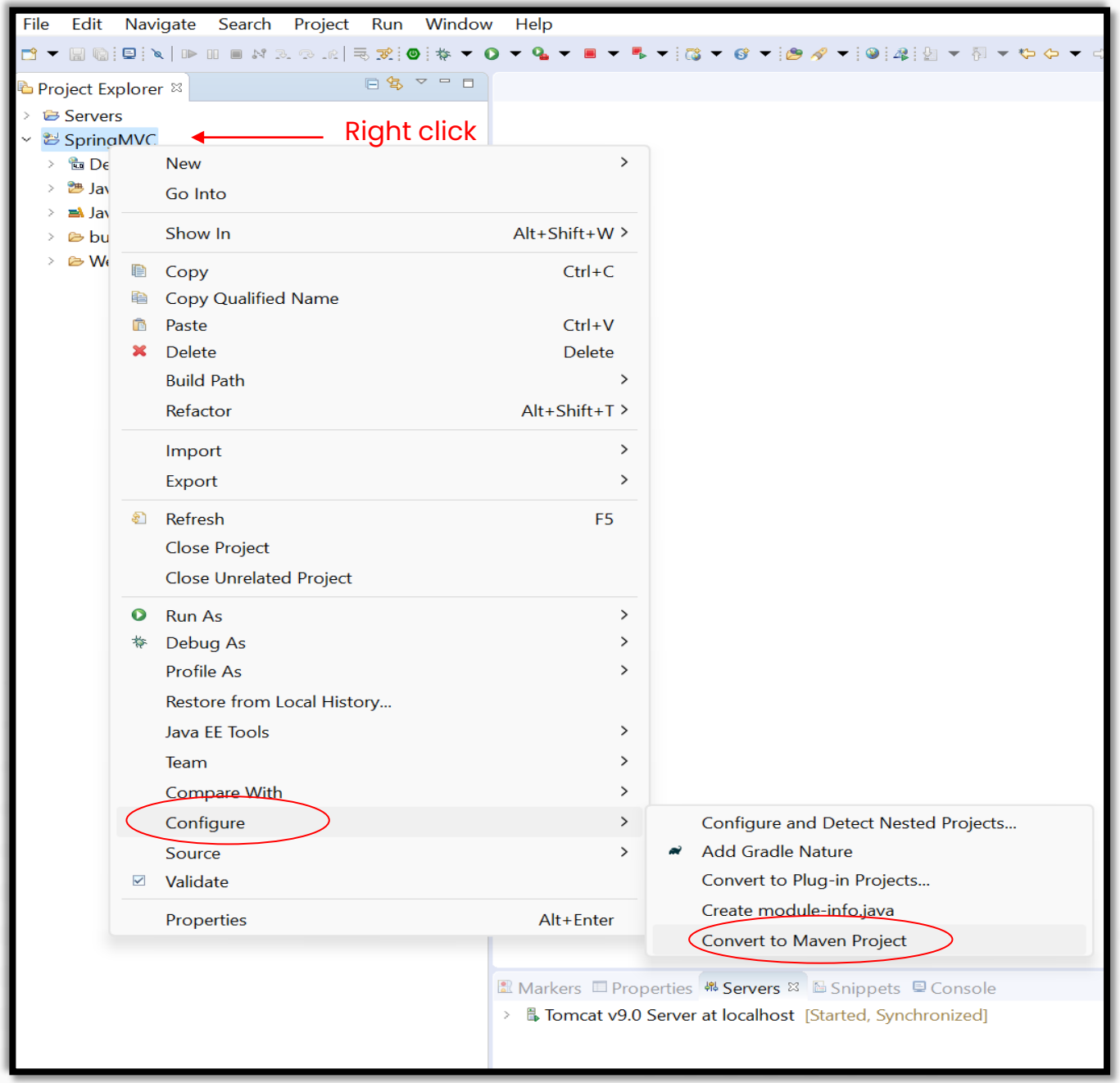
- Getting right JAR files for each project as there maybe different versions of separate packages.
- To download dependencies, visiting the official website of different software is not needed. We can refer to mvnrepository.com
- Helps to create the right project structure which is essential for execution.
- Building and deploying the project to make it works.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

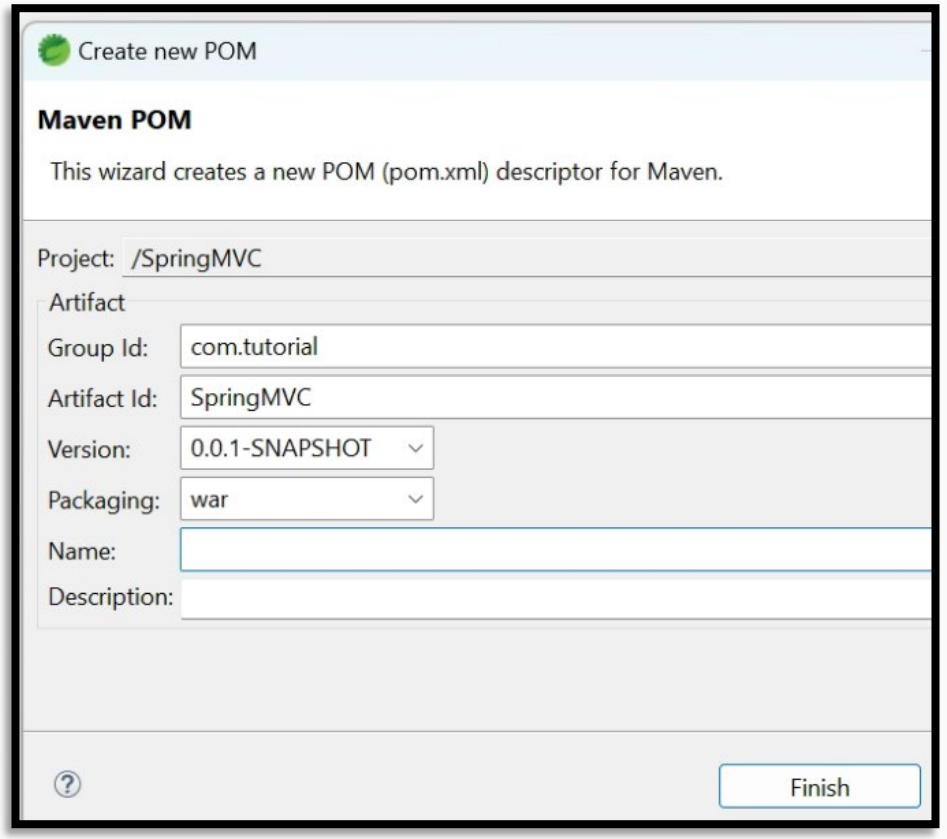
How to convert to Maven Project in STS?

Steps:

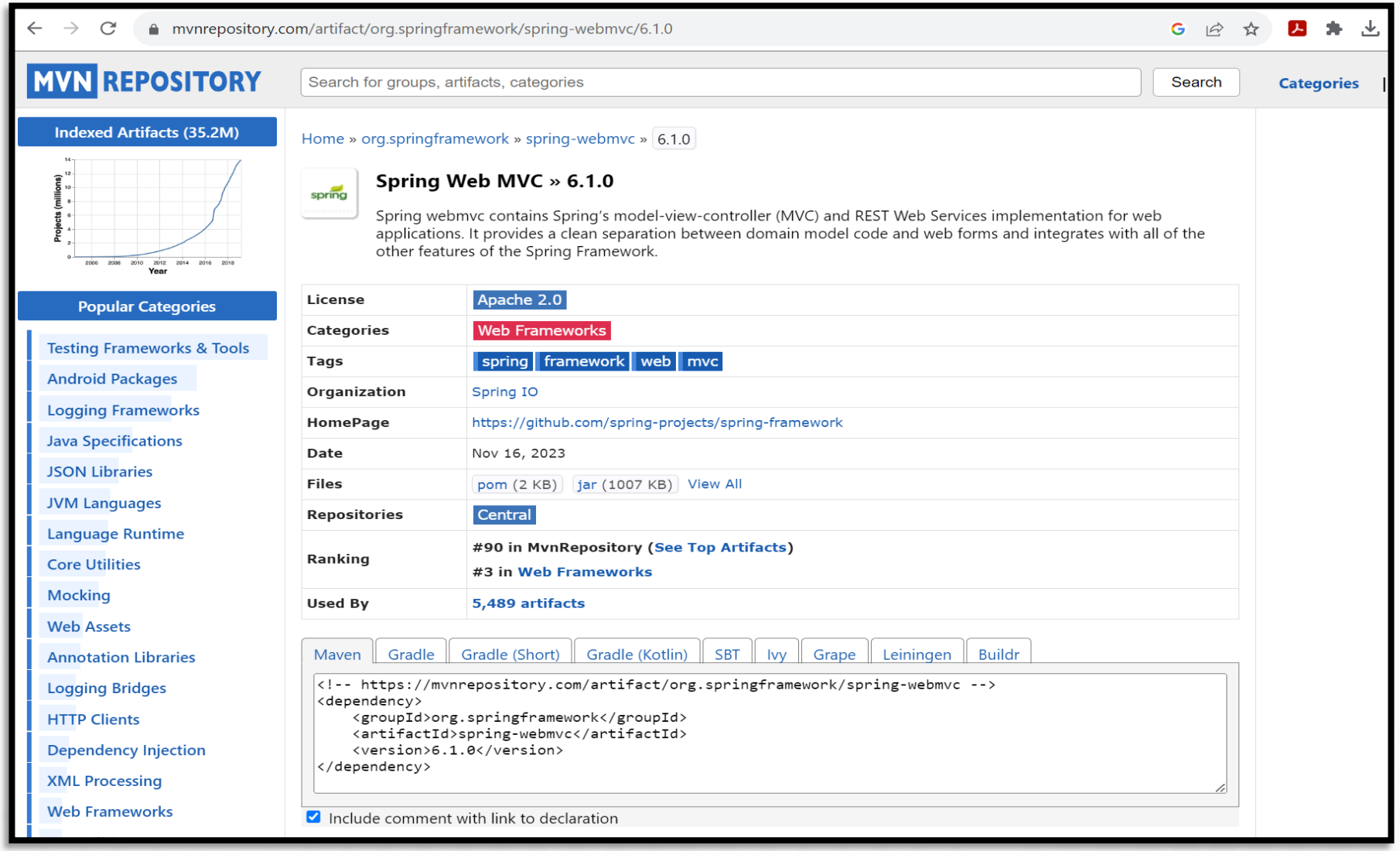
- Right click on your dynamic web project
- Select - Configure
- Select - Convert to Maven project



Group id : It uniquely identifies your project among all. Example: com.companyname
Artifact id : It is name of jar or war without version. It may be something like project.
Version : Version is used for version control for artifact id. If you distribute this project, you may incrementally create different version of it.



Platform – mvnrepository.com



Example of dependencies in pom.xml

If use previous version:
use Spring version 5

If use latest version:
use Spring version 6

If use latest version:
jakarta.servlet

```
<!--
https://mvnrepository.com/artifact/jakarta.servlet/
jakarta.servlet-api -->
<dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <version>6.0.0</version>
    <scope>provided</scope>
</dependency>
```

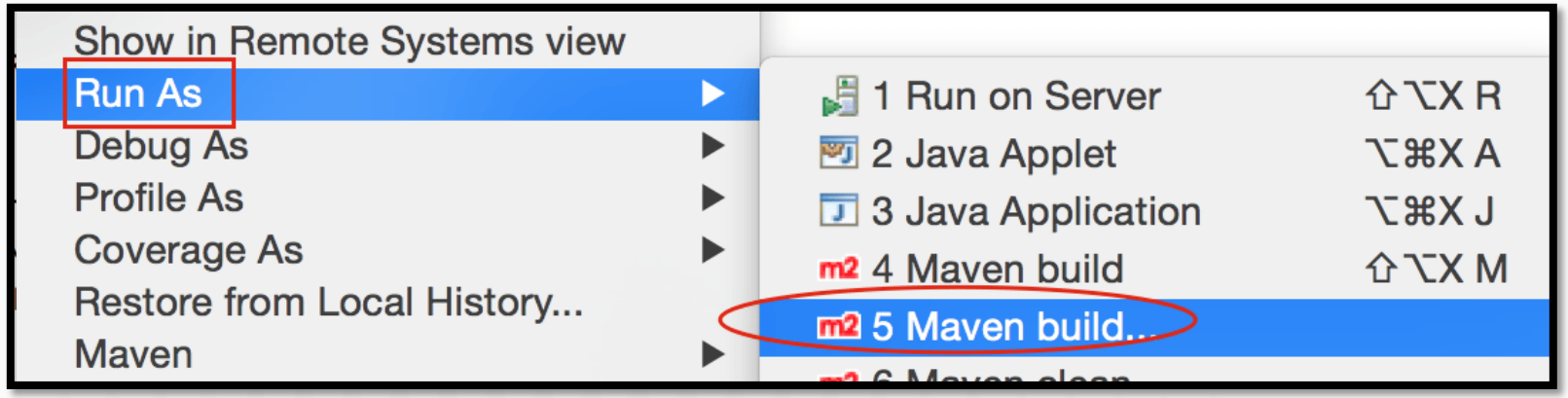
```
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.3.18</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>5.3.20</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.20</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-web -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>5.3.20</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

If use previous version:
javax.servlet

UTM UNIVERSITI TEKNOLOGI MALAYSIA

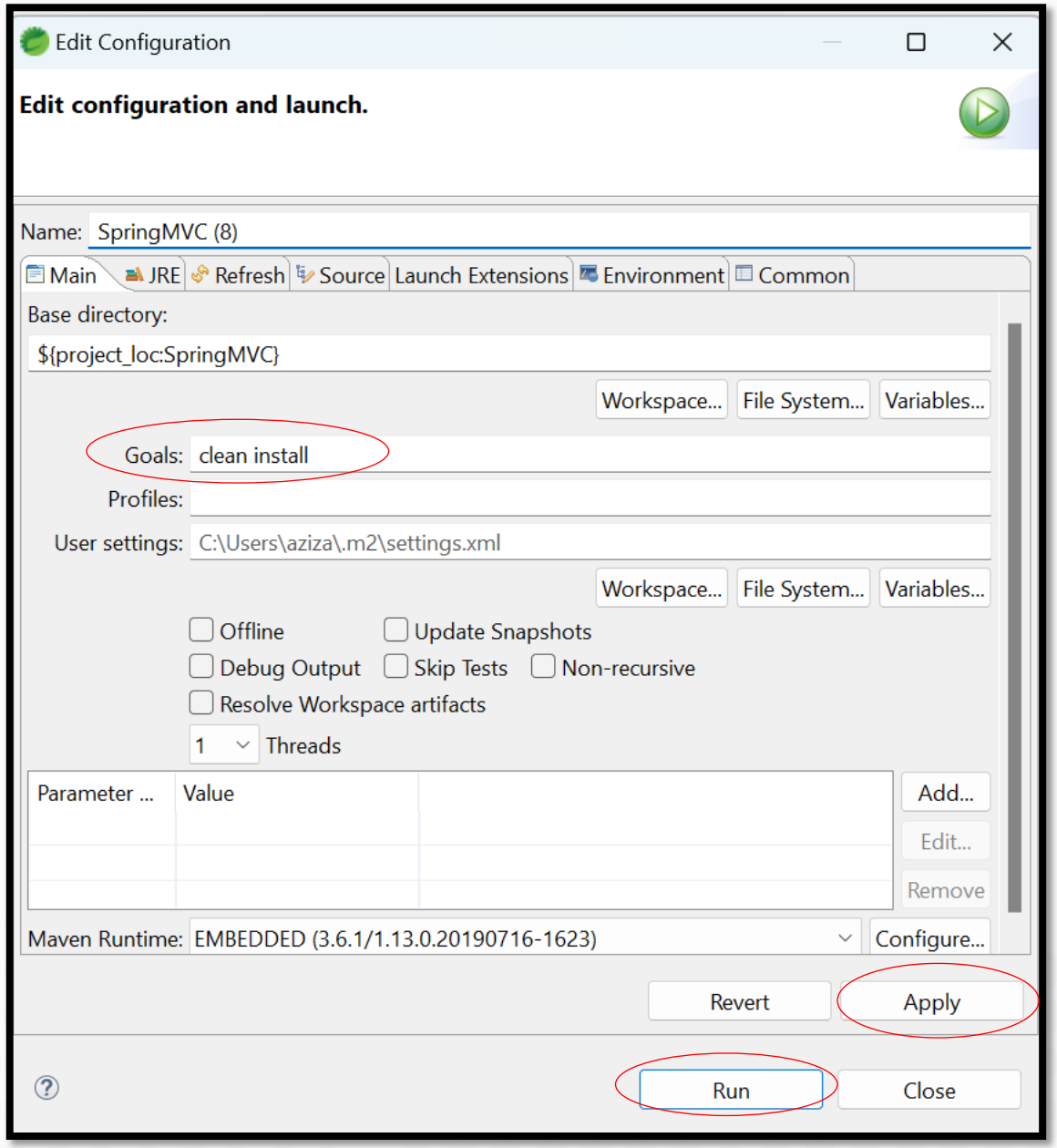
Run the Project

- To run the project:
  - Right click on your project
    - Select - Run As
    - Select - Maven build..



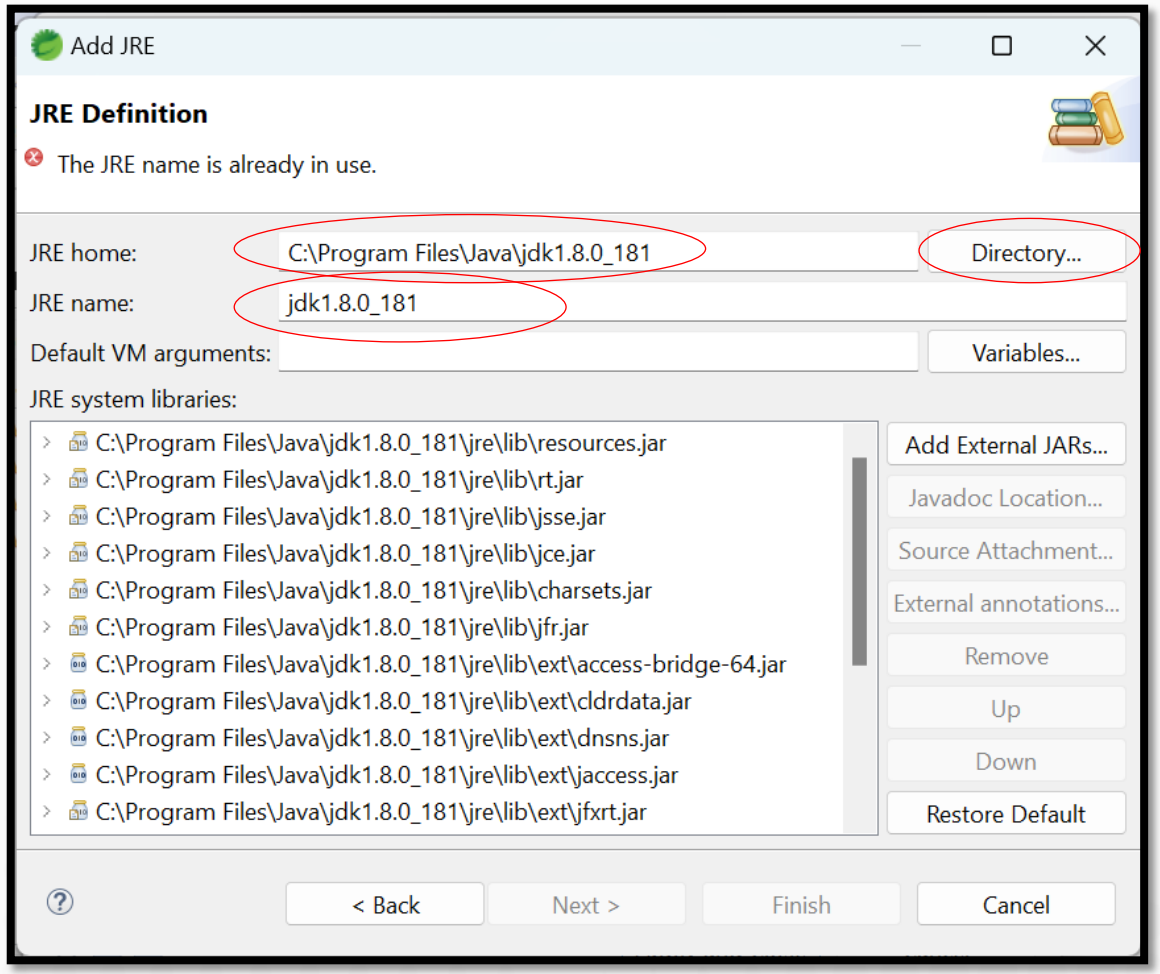
Run the Project

- Goals: Type - clean install
- Click - Apply
- Click - Run



How to fix if the Maven build is not success?

- Right Click on project
- -->Build Path
- --> Configure Build Path
- -->Add Library..
- -->JRE System Library --> Next
- --> Alternate JRE
- --> Installed JREs --> Add
- --> Standard VM --> Next
- --> Point to Java folder in C: drive (Windows) and select JDK folder and Finish.



Display the Output

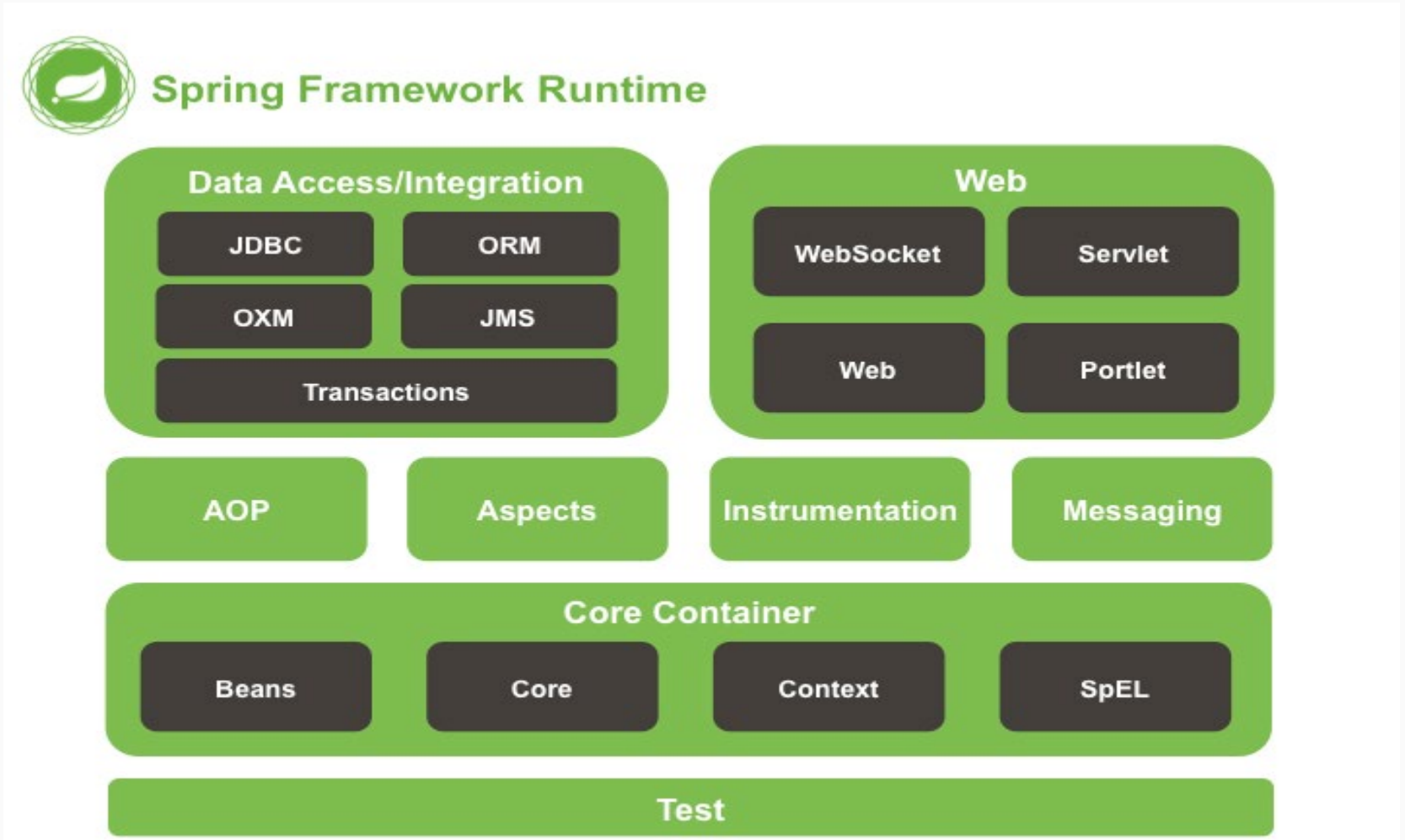
- To run the project:
  - ➢ Right click on your project
    - Select - Run As
    - Select - Run on Server
- Type in URL - http://localhost:8086/SpringMVC/welcome.html
- Example of output using Spring MVC:

Spring Framework

- Spring framework makes the easy development of JavaEE application.
- It is helpful for beginners and experienced persons.
- Spring is a *lightweight* framework.
- It can be thought of as a *framework of frameworks* because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF etc.
- The Spring Framework consists of features organized into about 20 modules. These modules are grouped into:
  - Core Container
  - Data Access/Integration
  - Web
  - AOP (Aspect Oriented Programming)
  - Instrumentation
  - Messaging
  - Test

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Spring Framework



Source: https://docs.spring.io/spring-framework/docs/4.3.x/spring-framework-reference/html/overview.html

Spring Framework

Core Feature:

- **Dependency Injection** (DI) is also called as **Inversion of Control** (IoC) is the most important feature of the Spring Framework and it is at the core of all Spring Modules.

Why is this DI or IoC so important?

- It is a software design patterns which is really useful for designing loosely coupled software components.
- These loosely coupled applications can easily be tested and maintained.
- It is easy to reuse your code in other applications.
- The dependencies won't be hard coded inside all java objects/classes, instead they will be defined in XML configuration files or configuration classes (Java Config).
- Spring Container is responsible for injecting dependencies of objects.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Advantages of Spring Framework

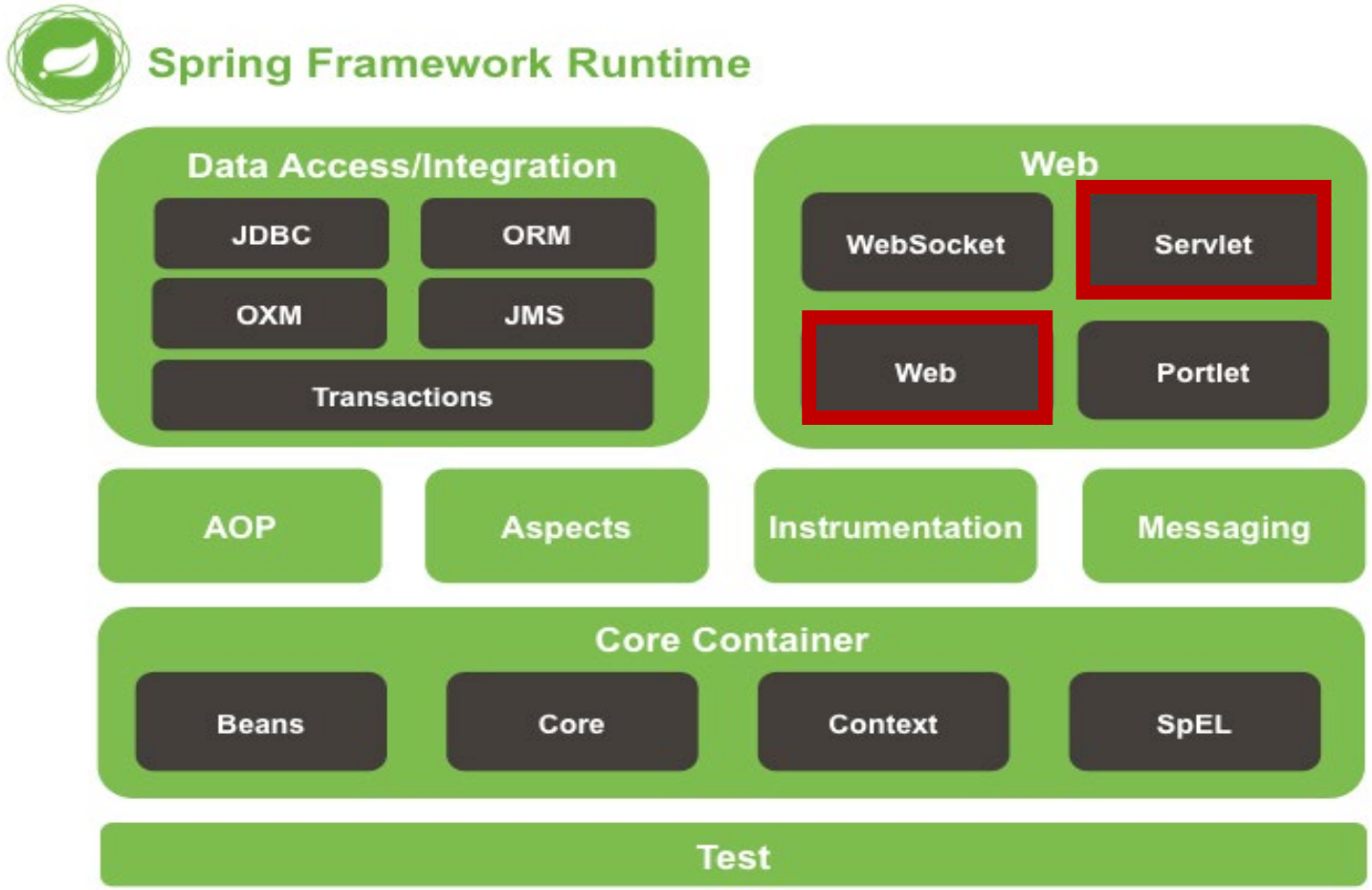
1) **Predefined templates**
   - Provides templates for JDBC, Hibernate, JPA etc. technologies. There is no need to write too much code. It hides the basic steps of these technologies.
2) **Loose coupling**
   - The Spring applications are loosely coupled because of dependency injection.
3) **Easy to test**
   - The Dependency Injection makes easier to test the application.
4) **Lightweight**
   - Lightweight because of its POJO implementation. It doesn't force the programmer to inherit any class or implement any interface.
5) **Fast development**
   - The Dependency Injection feature of Spring Framework and it support to

UTM UNIVERSITI TEKNOLOGI MALAYSIA

What is Spring MVC?

Module of Spring framework on Web layer:

- **Web Module** provides basic web-oriented integration features and the initialization of the IoC container using servlet listener and web application context.
- **Servlet Module** contains Spring MVC implementation for web application.



Source: https://docs.spring.io/spring-framework/docs/4.3.x/spring-framework-reference/html/overview.html

Spring MVC

- Spring MVC is an open source MVC 2 framework for developing Java web applications.
- It follows the Model-View-Controller design pattern.
- It implements all the basic features of a core spring framework like Inversion of Control and Dependency Injection.
- A Spring MVC provides a solution to use MVC in Spring framework by the help of **DispatcherServlet**.
- **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Spring MVC Components

- **DispatcherServlet** responsible for intercepting the request and dispatching for specific URL.
- **HandlerMapping** is an interface that defines a mapping between requests and handler objects. It determines which controller to call.
- **Controller** responsible for processing user requests and building appropriate model and passes it to the view for rendering.
- **ModelAndView** class object encapsulates view and model linking.
- **Model** encapsulates the application data, will consist of POJO.
- **ViewResolver** provides a mapping between view names and actual view.
- **View** interface represents presentation logic and is responsible for rendering content

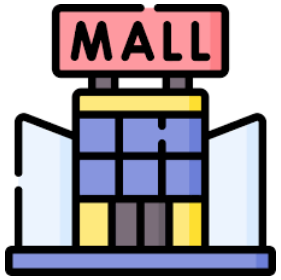
UTM UNIVERSITI TEKNOLOGI MALAYSIA

Spring Web MVC (Analogy)

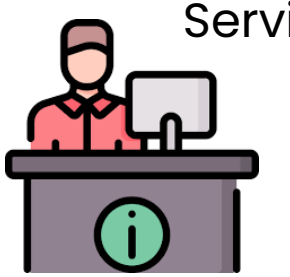
Customer

Request toy as a gift

Response toy as a gift



Customer Service



Spring Web MVC Flow

Request page 1

10 Response page

Handler Mapping

Get the URL and tell the exact address of the page that can generate data for the request

2 3 Dispatcher Servlet

Front Controller

4 5

Controller

Prepare the data, create a Java object to hold the data (it may take help from other Java classes if required)

Model And View

Java object holding the data and name of the final response builder

9 8

6 7

View

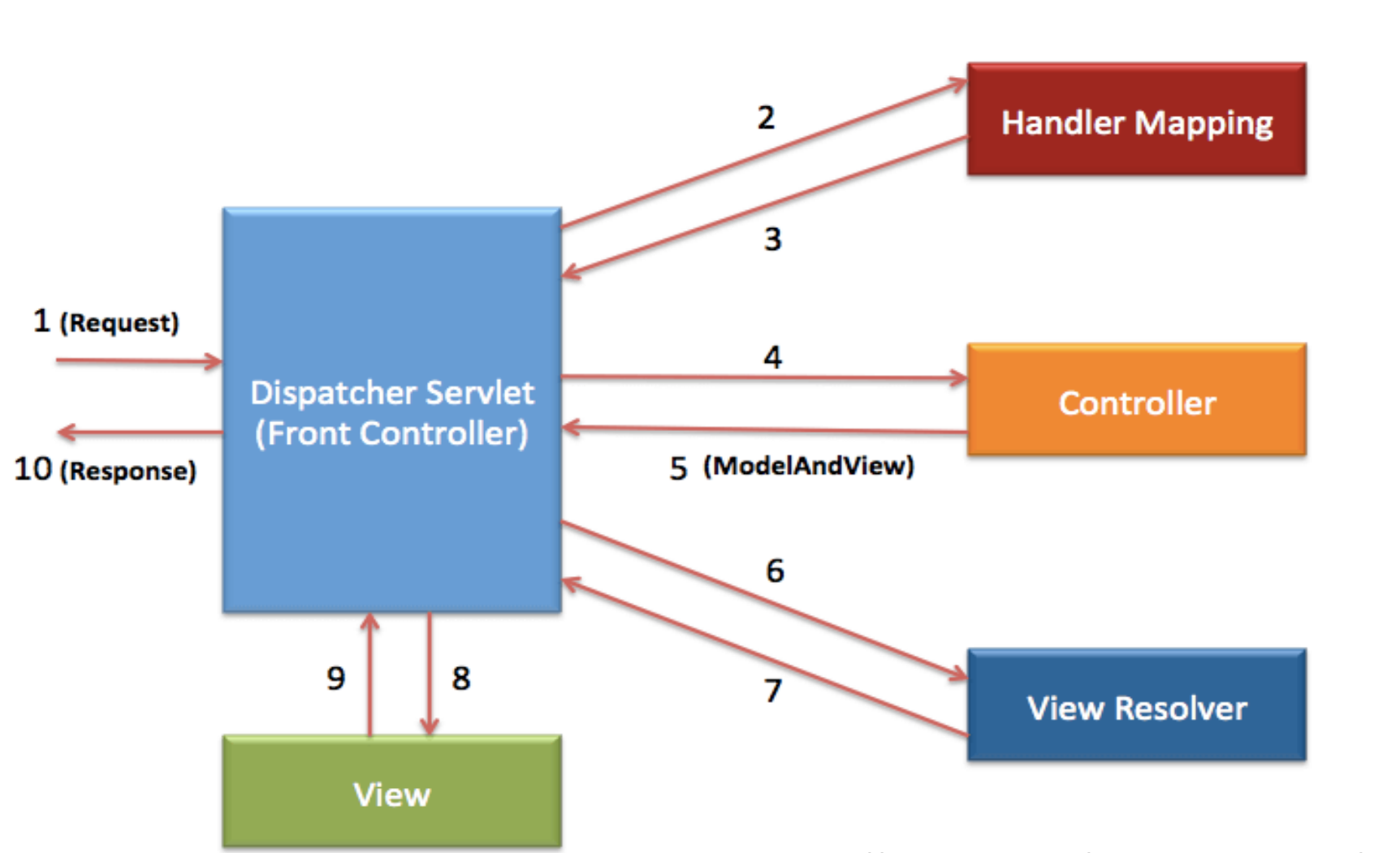
Display the actual view with the data

View Resolver

Mapping between view name and actual view (final response builder)

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Spring Web MVC Flow



Source: https://java2blog.com/spring-mvc-tutorial/

Spring Web MVC Flow

1. The request will be received by Front Controller i.e. **DispatcherServlet**.
2. DispatcherServlet will pass this request to HandlerMapping. **HandlerMapping** will find suitable Controller for the request.
3. **HandlerMapping** will send the details of the controller to DispatcherServlet.
4. DispatcherServlet will call the **Controller** identified by HandlerMapping. The **Controller** will process the request by calling appropriate method and prepare the data. It may call some business logic or directly retrieve data from the database.
5. The **Controller** will send **ModelAndView**(Model data and view name) to **DispatcherServlet**.
6. Once DispatcherServlet receives ModelAndView object, it will pass it to **ViewResolver** to find appropriate View.
7. **ViewResolver** will identify the view and send it back to **DispatcherServlet**.
8. **DispatcherServlet** will call appropriate **View** identified by ViewResolver.
9. The **View** will create Response in form of **HTML** and send it to **DispatcherServlet**.
10. **DispatcherServlet** will send the response to the **browser**. The browser will render the html code and display it to **end user**.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Front Controller – DispatcherServlet

- Spring MVC is designed around a central servlet named **DispatcherServlet**.
- Every request is handled by the DispatcherServlet.
- DispatcherServlet is also called a **Front Controller**.
- It uses to handle all incoming requests and uses customizable logic to determine which controllers should handle which requests.
- It forwards all responses to, through view handlers to determine the correct views to route responses.
- It exposes all beans defined in Spring to controllers for dependency injection.

Front Controller (Analogy)

Customer

Request cake

Response cake

MyCafé

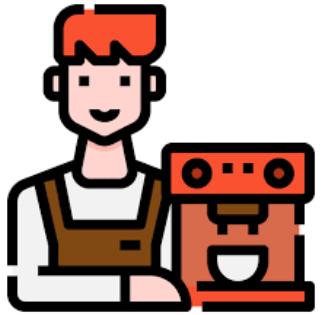
Waiter

Baker

cake

cake

Barista



coffee

coffee

Cashier

receipt

receipt

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Front Controller

Client

/myCafe/cake

cake

myCafe

MyCafé

Front Controller

/myCafe/*

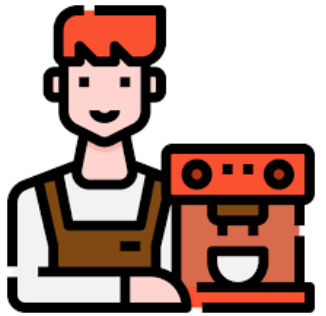
/cake

/coffee

/receipt

BakerController

BaristaController



PaymentController

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

DispatcherServlet in web.xml

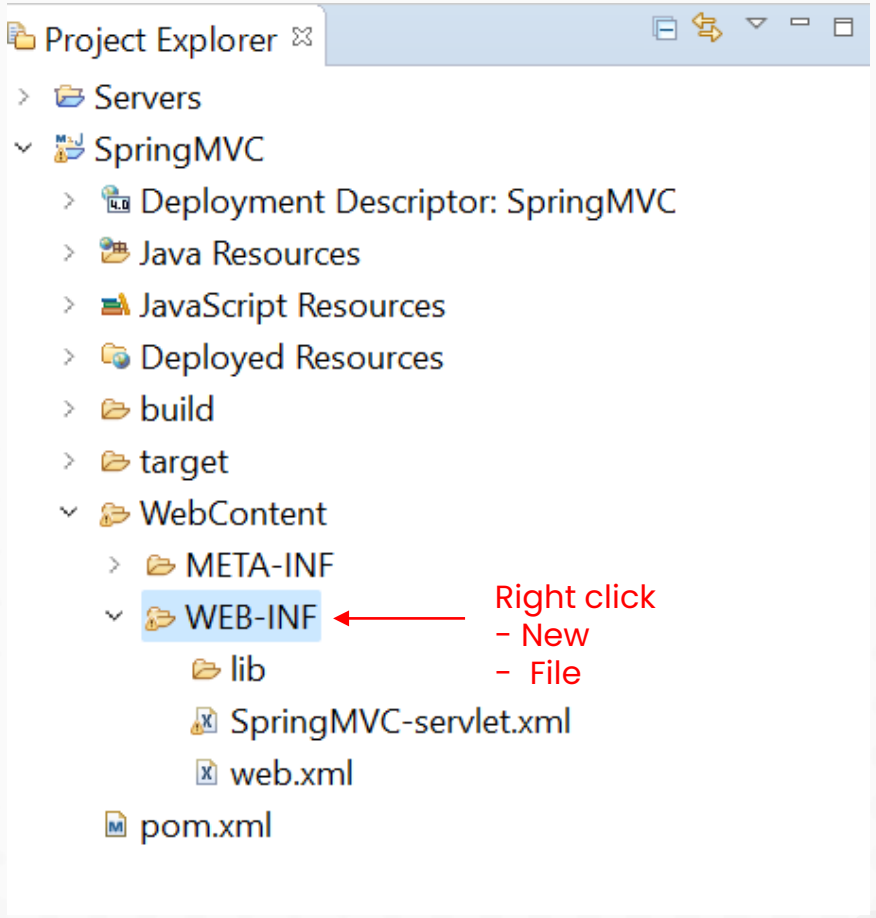
- **DispatcherServlet** is configured in order to dispatch incoming HTTP request to handlers and returns response to browsers.
- DispatcherServlet is configured in **web.xml** file.
- Example of DispatcherServlet in web.xml:

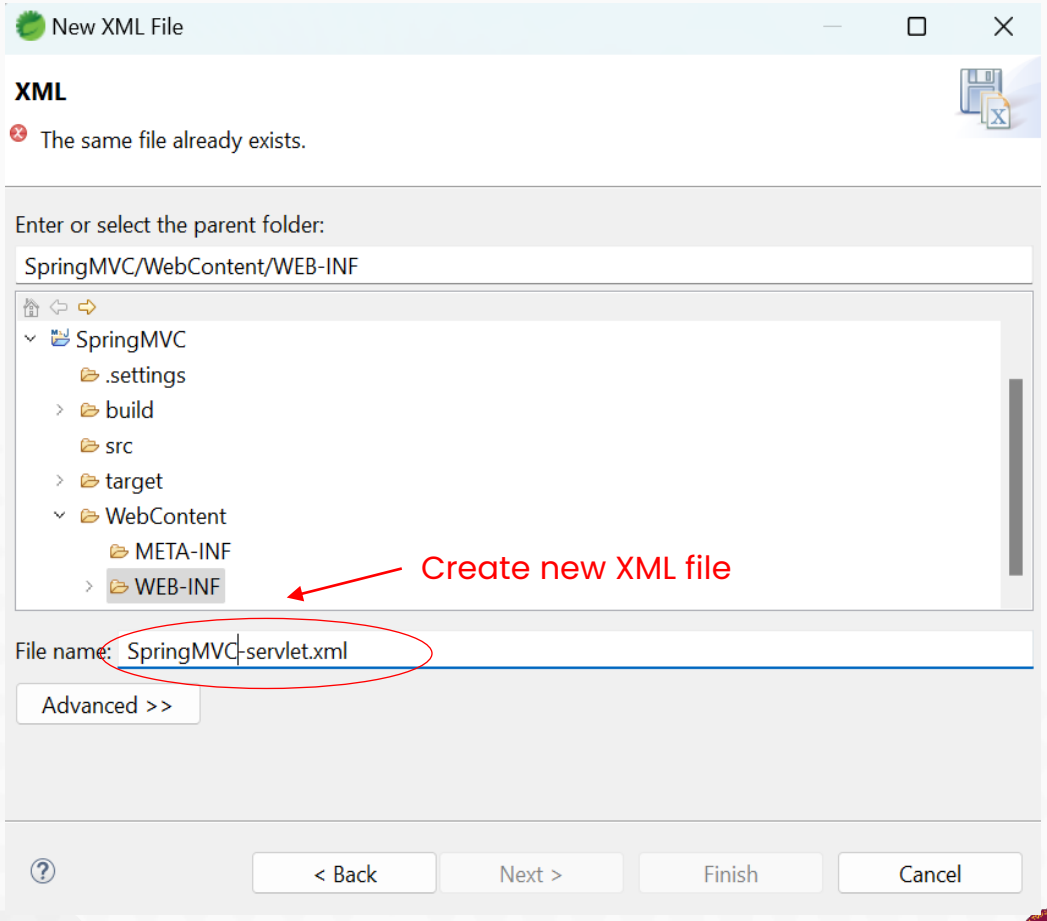
```
<servlet>
        <servlet-name>SpringMVC</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>SpringMVC</servlet-name>
        <url-pattern>/welcome.jsp</url-pattern>
        <url-pattern>/index.jsp</url-pattern>
        <url-pattern>/welcome.html</url-pattern>
        <url-pattern>*.html</url-pattern>
    </servlet-mapping>
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Spring Configuration

- By default, Spring looks for a **[servletname] – servlet.xml** file in the **/WEB-INF** folder to get the **DispatcherServlet** initialized.





Spring Configuration

- Example of **[servletname] – servlet.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="HandlerMapping"
class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

<bean name="/welcome.html" class="com.tutorial.controller.HelloController"/>

<bean id="viewResolver"
   class = "org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

Controller

- **DispatcherServlet** delegates the request to the controllers to execute the functionality specific to it.
- The @Controller annotation indicates that a particular class serves the role of a controller.
- Controller interpret user input and transform this input into specific model which will be represented to the user by the view.
- **@Controller** annotation defines the class as a Spring MVC controller.
- **@RequestMapping** annotation is used to map a URL to either an entire class or a particular handler method.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Controller – Without Annotation

- Example of **HelloController.java - controller** file:

```
package com.tutorial.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;

public class HelloController extends AbstractController {

    @Override
    protected ModelAndView handleRequestInternal (HttpServletRequest request,
    HttpServletResponse response) throws Exception{

    ModelAndView modelandview = new ModelAndView("HelloPage");
    modelandview.addObject("welcomeMessage", "Hi user, welcome to the first Spring MVC
    Application");

    return modelandview;
    }
}
```

Controller – With Annotation

- The **@Controller** annotation indicates that a particular class serves the role of a **controller**.
- Spring uses the **@RequestMapping** method annotation to define the URI Template for the request. It can be applied to class-level and/or method-level in a controller.
- Example of **HelloController.java - controller** file with @Controller and @Request Mapping annotation:

```
package com.tutorial.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HelloController {

    @RequestMapping("/welcome")
    protected ModelAndView helloWorld(){

        ModelAndView modelandview = new ModelAndView("HelloPage");
        modelandview.addObject("welcomeMessage", "Hello World with annotation! ");

        return modelandview;
    }
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Spring Configuration – With Annotation

- Example of **[servletname] – servlet.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="“
 http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
<bean id="HandlerMapping"
class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

<bean name="/welcome.html" class="com.tutorial.controller.HelloController"/>
<context:component-scan base-package="com.tutorial.controller" />

<bean id="viewResolver"
   class = "org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

MultiAction Controller

- MultiAction Controller is a Controller implementation that allows multiple request types to be handled by the same class.
- It means inside one controller class we can have many handler methods.

class level

method level

http://localhost:8086/SpringMVC/greet/welcome

http://localhost:8086/SpringMVC/greet/selamat

```
@Controller
@RequestMapping("/greet")
public class HelloController {

    @RequestMapping("/welcome")
    protected ModelAndView helloWorld(){

    ModelAndView modelandview = new ModelAndView("HelloPage");
    modelandview.addObject("welcomeMessage", "Hello World with annotation! ");

    return modelandview;
    }

    @RequestMapping("/selamat")
    protected ModelAndView selamatDatang(){

    ModelAndView modelandview = new ModelAndView("HelloPage");
    modelandview.addObject("welcomeMessage", "Selamat Datang with annotation! ");

    return modelandview;
    }

}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

@PathVariable Annotation

The **@PathVariable** annotation is used to extract the value of the template variables and assign their value to a method variable.

```
@Controller
public class HelloController {

      @RequestMapping("/welcome/{facultyName}/{userName}")
      protected ModelAndView helloWorld(@PathVariable ("facultyName") String faculty, @PathVariable ("userName") String name){

      ModelAndView modelandview = new ModelAndView("HelloPage");
      modelandview.addObject("welcomeMessage", "Hello " + name + " Your faculty is: " + faculty);

      return modelandview;
      }
}
```

User input

```
@Controller
public class HelloController {

      @RequestMapping("/welcome/{facultyName}/{userName}")
      protected ModelAndView helloWorld(@PathVariable Map<String,String> pathV ){

      String faculty = pathV.get("facultyName");
      String name = pathV.get("userName");

      ModelAndView modelandview = new ModelAndView("HelloPage");
      modelandview.addObject("welcomeMessage", "Hello " + name + "  Your faculty is: " + faculty);

      return modelandview;
      }
}
```

User input

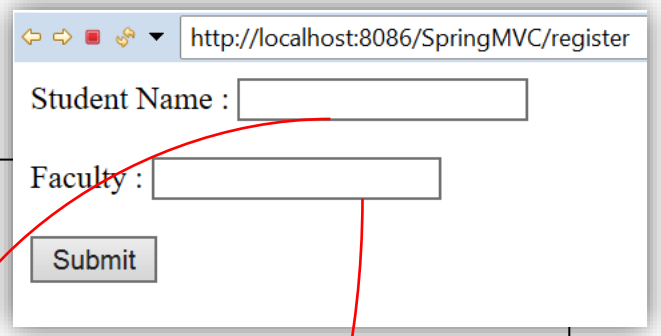
Same output with different way of using @PathVariable annotation in Map

Example of URL: http://localhost:8086/SpringMVC/welcome/FC/ali

@RequestParam Annotation

- **@RequestParam** is a Spring annotation used to bind a web request parameter to a method parameter.
- It has the following optional elements:
  - defaultValue - used as a fallback when the request parameter is not provided or has an empty value
  - name - name of the request parameter to bind to
  - required - tells whether the parameter is required
  - value - alias for name

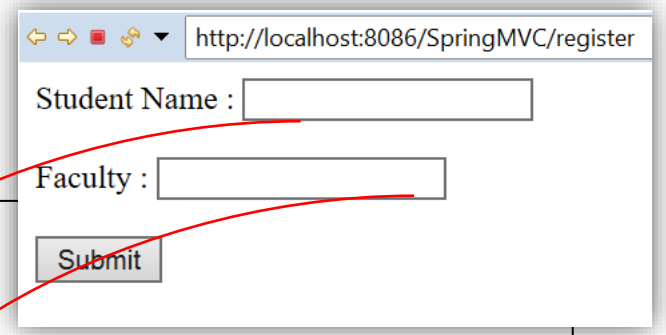
```
@Controller
public class RegisterController {

    @RequestMapping("/register")
    protected ModelAndView getRegisterForm( ){

        ModelAndView model = new ModelAndView("RegisterForm");

        return model;
    }

    @RequestMapping("/submit")
    protected ModelAndView submitRegisterForm(@RequestParam
    (“studentName”) String name, @RequestParam (“facultyName”)
    String faculty ){

        ModelAndView modelandview = new ModelAndView("HelloPage");
        modelandview.addObject("welcomeMessage", "Hello " + name +
        “, Your faculty is: " + faculty);

        return modelandview;
    }
}
```



@RequestParam Annotation

Same output with different way of using @RequestParam annotation in Map

```
@Controller
public class RegisterController {

    @RequestMapping("/register")
    protected ModelAndView getRegisterForm( ){

        ModelAndView model = new ModelAndView("RegisterForm");

        return model;
    }

    @RequestMapping("/submit")
    protected ModelAndView submitRegisterForm(@RequestParam Map<String,String> req ){

        String name = req.get("StudentName");
        String faculty = req.get("facultyName");

        ModelAndView modelandview = new ModelAndView("HelloPage");
        modelandview.addObject("welcomeMessage", "Hello " + name + ", Your faculty is: " + faculty);

        return modelandview;
    }
}
```



@ModelAttribute Annotation

- *@ModelAttribute* is an annotation that binds a method parameter or method return value to a named model attribute, and then exposes it to a web view.
- We can use *@ModelAttribute* either as a method parameter or at the method level.

Method level

```
@ModelAttribute
public void addAttributes(Model model) {
    model.addAttribute("msg", "Welcome to Malaysia!");
}
```

As a method parameter

```
@RequestMapping(value = "/addEmployee", method = RequestMethod.POST)
public String submit(@ModelAttribute("employee") Employee employee) {
    // Code that uses the employee object

    return "employeeView";
}
```

@ModelAttribute Annotation

Method level (without @ModelAttribute)

```
@RequestMapping("/register")
protected ModelAndView getRegisterForm() {

     ModelAndView model = new
     ModelAndView("RegisterForm");
     model.addObject("headerMessage", "Welcome to
     University");
     return model;
}

@RequestMapping("/submit")
protected ModelAndView submitRegisterForm
(@RequestParam ("studentName") String name,
@RequestParam ("facultyName") String faculty ){

     ModelAndView model = new
     ModelAndView("HelloPage");
     model.addObject("welcomeMessage", "Hello " +
     name + ", Your faculty is: " + faculty);
     model.addObject("headerMessage", "Welcome to
     University");
     return model;
}
```

Method level (with @ModelAttribute)

```
@RequestMapping("/register")
protected ModelAndView getRegisterForm() {

     ModelAndView model = new ModelAndView("RegisterForm");
     return model;
}

@RequestMapping("/submit")
protected ModelAndView submitRegisterForm (@RequestParam
("studentName") String name, @RequestParam ("facultyName")
String faculty ){
     ModelAndView model = new ModelAndView("HelloPage");
     model.addObject("welcomeMessage", "Hello " + name + ",
     Your faculty is: " + faculty);
     return model;
}

@ModelAttribute
public void addAttributes(Model model) {
     model.addAttribute("headerMessage", "Welcome to
     University");
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

@ModelAttribute Annotation

As a method parameter (without @ModelAttribute)

```
@RequestMapping("/submit")
protected ModelAndView submitRegisterForm(@RequestParam ("studentName") String name,
@RequestParam ("facultyName") String faculty ){

    Student student1 = new Student();
    student1.setStudentName(name);
    student1.setFacultyName(faculty);

    ModelAndView model = new ModelAndView("HelloPage");
    model.addObject("student1", student1);

    return model;
}
```

As a method parameter (with @ModelAttribute)

Reduce line of code

```
@RequestMapping("/submit")
protected ModelAndView submitRegisterForm(@ModelAttribute ("student1") Student
student1 ){

    ModelAndView model = new ModelAndView("HelloPage");
    return model;
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Model

- **Model** is generally defined as a MAP that can contain objects that are to be displayed in view.
- **ModelAndView** object encapsulates the relations between view and model and is returned by the corresponding Controller methods.
- **ModelAndView** class use **ModelMap** that is custom MAP implementation where values are added in key-value fashion.

ViewResolver

- The *ViewResolver* maps view names to actual views.
- Spring framework comes with quite a few view resolvers e.g. *InternalResourceViewResolver*, *BeanNameViewResolver,* and a few others.
- **InternalResourceViewResolver** is used to resolve the correct view based on **suffix** and **prefixes**, so the correct output (view) is resolved based on strings.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

ViewResolver

/myCafe/cake

Hello here is your cake!

Front Controller

/myCafe/*

Dispatcher Servlet

/cake

HelloPage

BakerController

```
@Controller
public class BakerController {
      @RequestMapping("/cake")
      protected ModelAndView
      requestCake(){
            ModelAndView modelandview =
            new
            ModelAndView("HelloPage");
            modelandview.addObject("msg",
            "Hello here is your cake! ");
            return modelandview;
            }
            }
```

HelloPage

View Resolver

WEB-INF/HelloPage.jsp

View

prefix

suffix

WEB-INF/ | HelloPage | .jsp

ViewResolver

```
<bean id="viewResolver"
   class =
"org.springframework.web.servlet.view.InternalResource
ViewResolver">
        <property name="prefix" value="/WEB-INF/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

View

- Java Spring MVC can be configured with different view technologies such as JSF,JSP, Velocity, Freemaker etc.
- **View** is responsible for displaying the content of Model objects on browsers as response output.
- View page can be explicitly returned as part of **ModelAndView** object by the controller.
- The view name can be independent of view technology and resolved to specific technology by using **ViewResolver** and rendered by **View**.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

View

- Example of **HelloPage.jsp - view** file:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"  %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>First Spring MVC</title>
</head>
     <body>
               <h1>First Spring MVC Application</h1>
               <h2>${welcomeMessage}</h2>
     </body>
</html>
```

Exception Handling: @ExceptionHandler Annotation

- Inside Spring MVC, users can define an exception handling class by implementing as below:

Example for Null Pointer Exception

```
@RequestMapping("/register")
protected ModelAndView getRegisterForm() throws Exception {

    String exceptionOccured = "NULL_POINTER";
    if (exceptionOccured.equalsIgnoreCase("NULL_POINTER")){
        throw new NullPointerException("Null Pointer Exception");
    }

    ModelAndView model = new ModelAndView("RegisterForm");
    return model;
}

@ExceptionHandler (value=NullPointerException.class)
public String handleNullPointerException(Exception e) {

//logging Null Pointer Exception
System.out.println ("Null Pointer Exception Occured: "+e);
return "NullPointerException";
}
```

Create a new JSP file with same name of the return value to display the output of error.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Exception Handling: @ControllerAdvice Annotation

Without @ControllerAdvice

```
//In part of RegisterController.java

@ExceptionHandler (value=NullPointerException.class)
public String handleNullPointerException(Exception e)
{

//logging Null Pointer Exception
System.out.println ("Null Pointer Exception Occured:
"+e);
return "NullPointerException";
}

@ExceptionHandler (value=IOException.class)
public String handleIOException(Exception e) {

//logging Null Pointer Exception
System.out.println ("IO Exception Occured: "+e);
return "IOException";
}
```

With @ControllerAdvice as Global Exception Handler

```
//In GlobalExceptionHandlerMethods.java

@ControllerAdvice
public class GlobalExceptionHandlerMethods {

    @ExceptionHandler (value=NullPointerException.class)
    public String handleNullPointerException(Exception e)
    {

    //logging Null Pointer Exception
    System.out.println ("Null Pointer Exception Occured:
    "+e);
    return "NullPointerException";
    }

    @ExceptionHandler (value=IOException.class)
    public String handleIOException(Exception e) {

    //logging Null Pointer Exception
    System.out.println ("IO Exception Occured: "+e);
    return "IOException";
    }
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

HTTP Messages

- 202 OK: Request successfully processed by the Server.
- 404 Not Found: Requested Resource is not available.
- 500 Internal Server Error: An unexpected condition/error occurred while processing a request.
- 503 Service Unavailable: The server is currently unavailable (because it is overloaded or down for maintenance).

- **Client Error 4xx
- **Server Error 5xx

- For complete list of Status code:

https://www.rfc-editor.org/rfc/rfc9110.html#name-client-error-4xx

UTM UNIVERSITI TEKNOLOGI MALAYSIA

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

TOPIC 7 – Spring Web MVC

The End

innovative • entrepreneurial • global

UTM JOHOR BAHRU

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SECJ 3303 – INTERNET PROGRAMMING

TOPIC 8 – USING RELATIONAL DATABASE

innovative • entrepreneurial • global

UTM JOHOR BAHRU

OBJECTIVES

Knowledge

- Understand the Role of Databases in Web Applications
- Overview of database connection methods.
- Traditional JDBC connection approach.
- Using non-PreparedStatement vs PreparedStatement for more secure database queries.
- Performing CRUD operations with standard JDBC.
- CRUD operations using PreparedStatement.
- Overview of Spring JDBC Template.
- Performing CRUD operations with Spring JDBC Template.
- Configure Hibernate in a web application.
- Map entities to database tables using annotations.
- Perform CRUD operations using Hibernate.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

OBJECTIVES

Applied

- Configure JDBC database connection.
- Implement basic CRUD operations.
- Apply parameterized queries for security.
- Set up data source and Spring JDBC Template.
- Set up Hibernate configurations.
- Handle autogenerated keys for identity columns.
- Implement CRUD operations with Hibernate.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Database in Web Application

In modern web development, the integration of databases is a cornerstone for managing data and enabling dynamic, data-driven applications

Importance of Databases in Web Development

- Data Storage and Retrieval: Databases serve as repositories for storing and retrieving data. They enable applications to persistently store information and retrieve it as needed for dynamic content presentation.

- Dynamic Content: Web applications often rely on databases to dynamically generate content based on user interactions, preferences, and real-time updates.

- User Authentication: Databases are integral for managing user accounts and authentication. User credentials and profiles are securely stored, allowing for secure access to applications.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Database in Web Application (Cont...)

Understanding the dynamic role of databases is essential for web developers. Whether opting for traditional relational databases or exploring cloud-based solutions like Firebase, developers have a range of tools to choose from based on the specific needs of their applications

Here we will cover 3 different approaches

1. **Using JDBC** - (Java Database Connectivity), which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
2. **Using Spring JDBC Template -** focuses on the Spring JDBC Template, a higher-level abstraction that simplifies database operations within the Spring framework.
3. **Using Hibernate -** an ORM framework.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Using Relational Database in Web Application

Part 1:

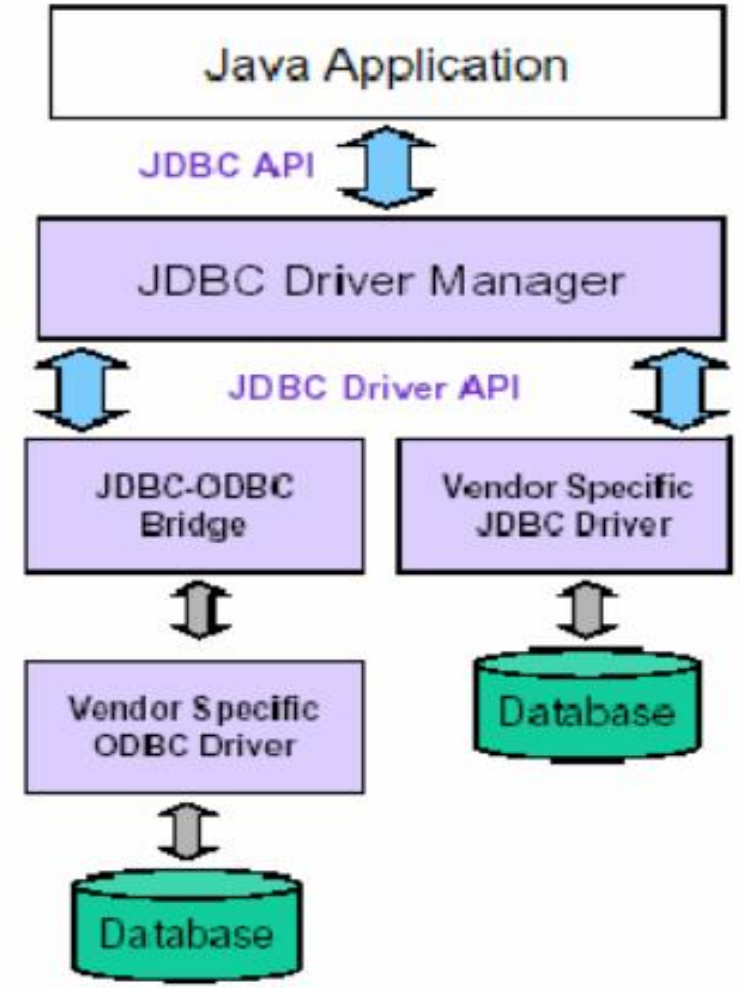
Using JDBC - (Java Database Connectivity), which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

JDBC serves as a bridge between Java applications and relational databases, enabling seamless communication, data retrieval, and manipulation.

JDBC simplifies the process of working with databases in Java programs and provides a standardized way to interact with different database management systems

UTM UNIVERSITI TEKNOLOGI MALAYSIA

JDBC General Architecture



Preparations

Prep 1: Use your existing spring web project
(recent spring web project from topic 8 - Spring Web)
Add dependency to project's pom.xml

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.23</version>
</dependency>
```

Prep 2: mysql database, with name="springdb",
2 tables : i. Customer (id, name, address, contactNum, email)
ii. Product (id, name, price, quantity)

Prep 3 : insert several records to table Customer and Product

UTM UNIVERSITI TEKNOLOGI MALAYSIA

The 7 essential JDBC steps for Java database interaction

1. Load the Driver
2. Establish a connection
3. Create JDBC Statements
4. Execute SQL Statements
5. GET ResultSet
6. Close statement
7. Close connection

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Step 1: Load the Driver

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

Step 2 : Establish a connection

```
Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/springdb",
"root", "");
```


UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Step 3: Create statement or preparedStatement

```
Statement stmt = Connection.createStatement();
```

Step 4 : Execute SQL Statement/PreparedStatement

Step 5 : Get ResultSet, and iterate through the resultset

```
ResultSet rs = stmt.executeQuery("SELECT * FROM PRODUCT")) {
        while (rs.next()) {
          //process the resultset
          }
```

Step 6 & 7 : Close Statement and Connection objects

```
stmt.close();
conn.close()
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Updating Data Through Transactions

Transactions ensure the atomicity, consistency, isolation, and durability (ACID) properties of database operations

code example:

```
try {
    connection.setAutoCommit(false);
    //Perform multiple SQL operations as one atomic transaction unit
    //For example, update the quantity in the 'product' table for the
    //specified product ID and simultaneously add a new order to the
    //'Order' table
    connection.commit();
} catch (SQLException e) {
    connection.rollback();
} finally {
    connection.setAutoCommit(true);
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Implementing Step 1 - Step 7 in ProductDAOJdbc.java

```java
//code segment for ProductDAOJdbc.java
@Service
public class ProductDAOJdbc {
    private static final String JDBC_URL = "jdbc:mysql://localhost:3306/springdb";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    private Connection getConnection() {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(JDBC_URL, USER, PASSWORD);
        } catch (Exception ex) {
            ex.printStackTrace();
            throw new RuntimeException("Error establishing database connection", ex);
        }
    }
    public void add(Product product) {
        try (Connection connection = getConnection()) {
            String insertQuery = "INSERT INTO PRODUCT (name, price, quantity) VALUES (?, ?, ?)";
            try (PreparedStatement ps = connection.prepareStatement(insertQuery)) {
                ps.setString(1, product.getName());
                ps.setDouble(2, product.getPrice());
                ps.setInt(3, product.getQuantity());
                ps.executeUpdate();
                System.out.println("Product inserted successfully!");
            }
        } catch (Exception ex) {
            ex.printStackTrace();}
    }
```

Implementing Step 1 - 7 in ProductDAOJdbc.java (cont...)

```
44      //UPDATE OPERATION
45      public boolean update(String name) {
46          // complete the implementation here
47          return boolean(affectedRow!=0);
48      }
49
50      //DELETE OPERATION
51      public boolean delete(String name) {
52          // complete the implementation here
53          return boolean(affetedRow!=0);
54      }
55
56      //READ-SINGLE OPERATION
57      public Product getById(int id) {
58          Product product;
59          //complete the implementation here
60          return product;
61      }
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Implementing Step 1 - 7 in ProductDAOJdbc.java (cont...)

```
56      //READ-SINGLE OPERATION
57      public Product getById(int id) {
58          Product product;
59          //complete the implementation here
60          return product;
61      }
62      //READ-ALL OPERATION
63      public List<Product> getAll() {
64          List<Product> productList = new ArrayList<>();
65
66          try (Connection connection = getConnection();
67               Statement stmt = connection.createStatement();
68               ResultSet rs = stmt.executeQuery("SELECT * FROM PRODUCT")) {
69
70              while (rs.next()) {
71                  Product product = new Product(rs.getString(2), rs.getDouble(3), rs.getInt(4));
72                  productList.add(product);
73              }
74          } catch (Exception ex) {
75              ex.printStackTrace();  }
76          return productList;
77      }
78  }// end class ProductDAOJdbc
```

ProductController invokes ProductDAOJdbc

```
17 @Controller
18 @RequestMapping("/product")
19 public class ProductController {
20     private final ProductDAOJdbc productDaoJdbc;
21
22     //dependency injection in constructor
23     public ProductController(ProductDAOJdbc productDaoJdbc) {
24         this.productDaoJdbc = productDaoJdbc;
25     }
26
27     @GetMapping("/tryCreate")
28     public ModelAndView tryCreate() {
29         Product p = new Product("Scalp Care Shampoo",25.50, 10);
30         productDaoJdbc.add(p);
31         ModelAndView mav = new ModelAndView("created");
32         mav.addObject("p",p);
33         return mav;
34     }
35
36     @GetMapping("/tryGetall")
37     public String tryGetall(Model model) {
38         List<Product> prodList = productDaoJdbc.getAll();
39         model.addAttribute("prodList",prodList);
40         return "allproduct";
41     }
42     //complete the remaining request patterns
43 }//end class ProductController
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Using Relational Database in Web Application

Part 2:

Using Spring JDBC Template - focuses on the Spring JDBC Template, a higher-level abstraction that simplifies database operations within the Spring framework.

OVERVIEW

Spring JDBC allows developers to focus on core logic. Spring handles all the low level details and repetitive tasks like opening and closing of connections, exceptions, iteration of resultsets, transactions etc.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

SPRING JDBC – WHO DOES WHAT?

Following table shows what actions spring take care of and which actions are responsibility of application developer

Action	Spring	You
Define connection parameters.		X
Open the connection.	X	
Specify the SQL statement.		X
Declare parameters and provide parameter values		X
Prepare and execute the statement.	X	
Set up the loop to iterate through the results (if any).	X	
Do the work for each iteration.		X
Process any exception.	X	
Handle transactions.	X	
Close the connection, statement and resultset.	X	

UTM UNIVERSITI TEKNOLOGI MALAYSIA

cont...

Step 1: add dependency in pom.xml :

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.2.25.RELEASE</version>
</dependency>
```

Step 2: Create a configuration class, to set up the **DataSource** and **JdbcTemplate**

The **DatabaseConfig.java** is annotated with ***@Configuration***, indicating that it contains **bean** definitions. It defines **two beans**:

1. dataSource(): Configures a DriverManagerDataSource bean, which is a basic DataSource implementation provided by Spring.

2. jdbcTemplate(DataSource dataSource): Configures a JdbcTemplate bean, injecting the DataSource bean created in the previous step. Then, this JdbcTemplate bean can be used to execute the SQL queries to the database.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

```java
//partial code for DatabaseConfig.java
package service;
//import statements here

@Configuration
public class DatabaseConfig {
        private DriverManagerDataSource dataSource;

    @Bean
    public DataSource dataSource() {
        this.dataSource = new DriverManagerDataSource();
        //in real scenario, put the following hard-coded values into file ie, jdbcConfig.properties
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://localhost:3306/springdb");
        dataSource.setUsername("root");
        dataSource.setPassword("");
        return dataSource;
    }

    @Bean
    @Autowired      //dependency injection using constructor
    public JdbcTemplate jdbcTemplate(DataSource dataSource) {
        return new JdbcTemplate(dataSource);
    }
    //you may add code for close() and cleanUp() to properly close connections (omitted for brevity)
} //end class DatabaseConfig
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

cont...

Step 3: use the instance of JdbcTemplate, to invoke related methods for executing queries

class: JdbcTemplate (org.springframework.jdbc.core.JdbcTemplate): The core class of Spring JDBC framework, simplifies the use of JDBC and helps avoid common errors. It provides various methods for executing SQL queries, updates, and other operations.

important methods:

1. query: public <T> List<T> query(String sql, RowMapper<T> rowMapper, Object... args) throws DataAccessException

Executes a query and maps each row to an object using the provided RowMapper. Returns a list of objects.

example:

```
String sql = "select name, address, contactNum, email from customer";
List<Customer> custList = jdbcTemplate.query(sql, new BeanPropertyRowMapper(Customer.class));
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

cont...

important methods (cont...):

2. update: public int update(String sql, Object... args) throws DataAccessException

Executes an SQL update, insert, or delete statement. Returns the number of affected rows.

example:

```
String sql = "INSERT INTO CUSTOMER (name, address, contactNum, email) values (?,?,?,?)";
Object args[] = {"ali","taman","1234567","email@gmail.com"};
jdbcTemplate.update(sql,args);
```

3. queryForObject:

public <T> T queryForObject(String sql, Class<T> requiredType, Object... args) throws DataAccessException

Executes a query and returns a single result as an object of the specified type.

example:

```
String sql = "SELECT * FROM CUSTOMER WHERE ID=?";
Customer cust = jdbcTemplate.queryForObject(sql,
        new BeanPropertyRowMapper<Customer>(Customer.class), id);
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

cont...

important methods (cont...):

4. queryForList:

public List<Map<String, Object>> queryForList(String sql, Object... args) throws DataAccessException

Executes a query and returns the results as a list of maps, where each map represents a row with column names and values.

5. batchUpdate

public int[] batchUpdate(String sql, List<Object[]> batchArgs) throws DataAccessException

Executes a batch of SQL updates. Takes a list of parameter arrays, each representing the parameters for a single update.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

cont...

class BeanPropertyRowMapper : org.springframework.jdbc.core.BeanPropertyRowMapper
Maps rows to objects by matching the column names to the properties of the target class. It simplifies the mapping process, especially when the column names in the ResultSet match the field names in the Java class.

example use:

```
String sql = "select name, address, contactNum, email from customer";
List<Customer> custList = jdbcTemplate.query(sql, new BeanPropertyRowMapper(Customer.class));
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

```
// partial code for CustomerDAO.java
//import statements here

@Service
public class CustomerDAO {
   private JdbcTemplate jdbcTemplate;

   @Autowired       //dependency injection using constructor
   public CustomerDAO(JdbcTemplate jdbcTemplate) {
      this.jdbcTemplate = jdbcTemplate;  //reference to bean instance in IoC container
   }
   //IMPLEMENTATION OF CRUD OPERATIONS HERE
   //CREATE
   public String add() {
            String sql = "INSERT INTO CUSTOMER (name, address, contactNum, email) values (?,?,?,?)";
            Object args[] = {"ali","taman","1234567","email@gmail.com"};
            jdbcTemplate.update(sql,args);
            return "success";
   }
   //READ-ALL
   public List<Customer> getall(){
            String sql = "select name, address, contactNum, email from customer";
            List<Customer> custList = jdbcTemplate.query(sql, new BeanPropertyRowMapper(Customer.class));
            return custList;
   }//end getall
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

```
//cont....
//READ ie BY-NAME (this may return multiple rows)
public List<Customer> getByName(String name) {

            String sql = "SELECT * FROM CUSTOMER WHERE NAME=?";
            List<Customer> custList = jdbcTemplate.query(sql,
                  new    BeanPropertyRowMapper <Customer>(Customer.class), name);
            return custList;
      } //end getByName

    //impl more READ methods ie READ-BY-ID (this return single row)

   //impl UPDATE ie BY-ID

   //impl DELETE ie BY-ID

}//end CustomerDAO
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

Step 4: Invoke the CustomerDAO from the Controller class

```
19 //code for CustomerController.java
20 @Controller
21 @RequestMapping("/customer")
22 public class CustomerController {
23
24     @Autowired
25     private CustomerDAO custDao;
26
27     public CustomerController() {
28     }
29
30     @GetMapping("/getall")
31     public ModelAndView getall() {
32         List<Customer> custList = custDao.getall();
33         ModelAndView mav = new ModelAndView("customerList");
34         mav.addObject("custList",custList);
35     return mav;
36     }
37
38     @ResponseBody
39     @GetMapping("/getById")
40     public String getById(@RequestParam("id") int id, Model model) {
41         Customer cust = custDao.getById(id);
42         model.addAttribute("cust",cust);
43         return  "viewCustById";
44     }
45     //more codes here
```


UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Using Relational Database in Web Application

Part 3:

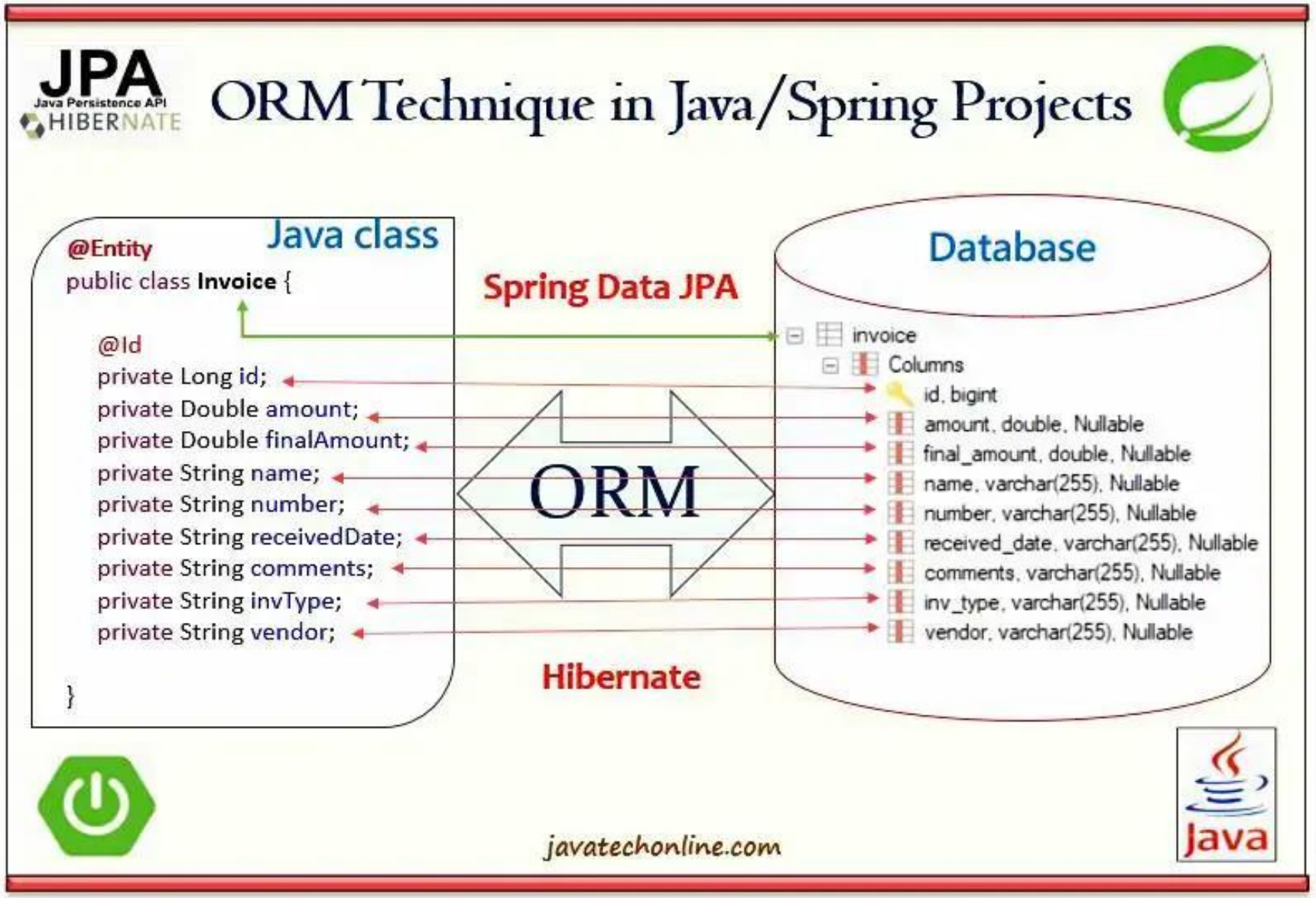
Using Hibernate - is an open source java based library framework. As an ORM framework, it allows the mapping of the Java domain object with database tables and vice versa.

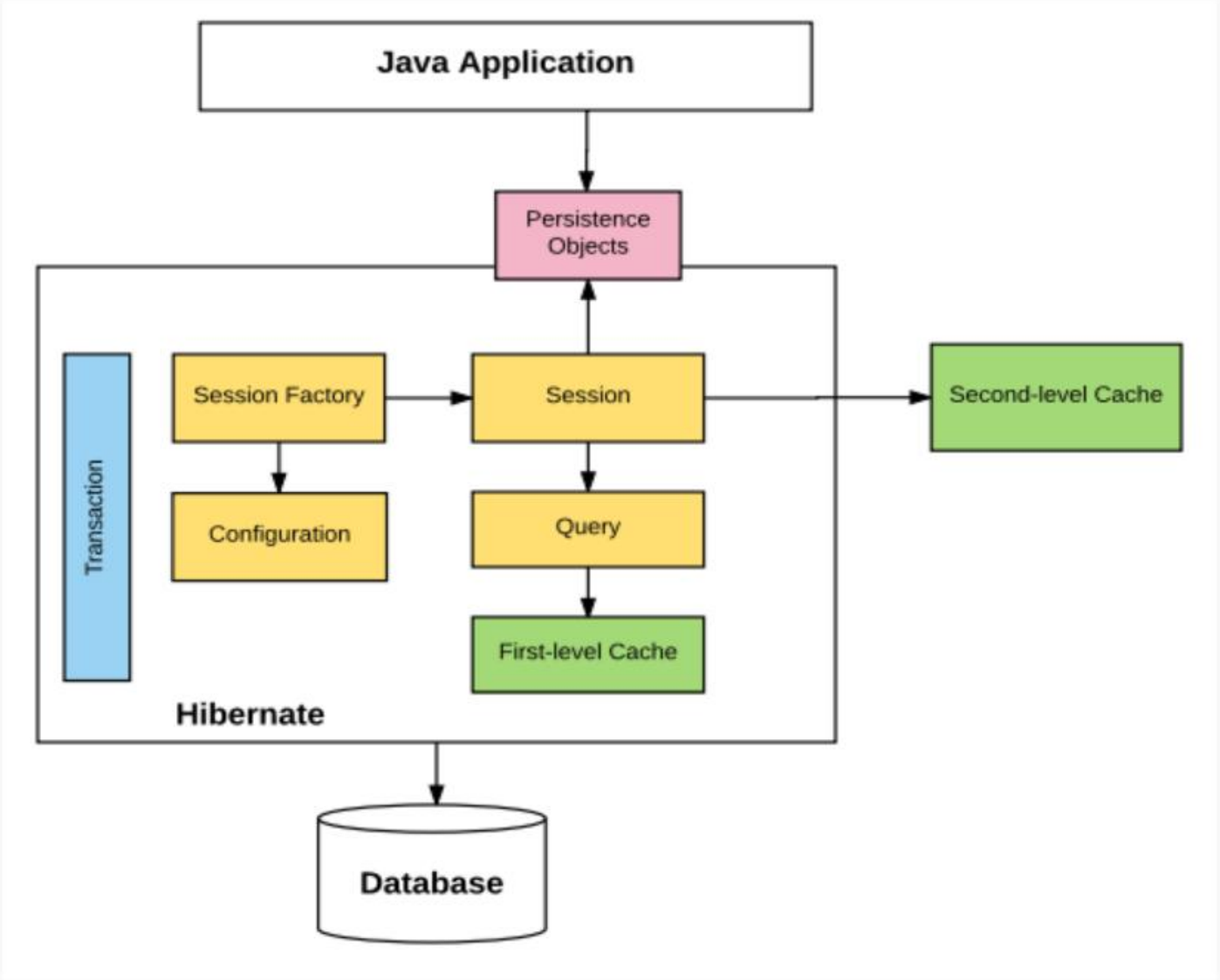
Using Hibernate, business logic is able to access and manipulate database entities via Java objects.

It helps to speed up the overall development process by taking care of aspects such as transaction management, automatic primary key generation, managing database connections and related implementations, and so on.

Implements Java Persistent API (JPA), which is a specification of Java which is used to access, manage, and persist data between Java object and relational database. It is considered as a standard approach for Object Relational Mapping.

UTM UNIVERSITI TEKNOLOGI MALAYSIA


JPA
Java Persistence API
HIBERNATE
ORM Technique in Java/Spring Projects
Java class
@Entity
public class Invoice {
@Id
private Long id;
private Double amount;
private Double finalAmount;
private String name;
private String number;
private String receivedDate;
private String comments;
private String invType;
private String vendor;
}
Spring Data JPA
ORM
Hibernate
Database
invoice
Columns
id, bigint
amount, double, Nullable
final_amount, double, Nullable
name, varchar(255), Nullable
number, varchar(255), Nullable
received_date, varchar(255), Nullable
comments, varchar(255), Nullable
inv_type, varchar(255), Nullable
vendor, varchar(255), Nullable
javatechonline.com
Java



Hibernate Architecture

Hibernate Architecture Components

:Hibernate follows a layered architecture comprising three main components: **Configuration, SessionFactory**, and **Session**

1. Configuration:

The Configuration component is responsible for configuring Hibernate and creating a representation of the application's domain model.

Role:

Sets up Hibernate by reading configuration files, mapping documents, and creating a configuration instance.

2. SessionFactory:

The SessionFactory is a factory of session objects and is a heavyweight object.

Role:

Acts as a factory for creating Session instances. It is thread-safe and should be instantiated only once in the application.

Responsibilities:

Manages Hibernate's configuration information.

Provides a convenient mechanism for opening sessions.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

3. Session:

Session is a lightweight object that represents a single-threaded unit of work.

Role:

Represents a connection to the database and provides CRUD operations. It is created by the SessionFactory.

Responsibilities:

Wraps a JDBC connection.

Performs database operations (save, update, delete, query).

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Roles and Relationships

Configuration: Prepares Hibernate for use by setting up the environment and mapping files.
SessionFactory: Acts as a factory to create Session instances, manages configuration, and maintains a cache of data, optimizing performance.
Session: Represents a single unit of work and provides methods for interacting with the database. Manages transactional behavior.

Interaction Flow:

Configuration: Initializes Hibernate settings.
SessionFactory: Creates a Session based on the configured settings.
Session: Performs database operations and manages the transactional context.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Transaction Management, HQL & Association Mapping

Transaction Management: Hibernate also supports transactions, allowing developers to group a set of operations into a single unit of work. Transactions ensure the consistency of the database by either committing or rolling back changes.

Hibernate Query Language (HQL): is an object-oriented query language, similar to SQL but using Java objects instead of database tables.

Association Mapping: Hibernate allows developers to define relationships between entities using associations such as one-to-one, one-to-many, many-to-one, and many-to-many. This helps in modeling complex relationships in the database

UTM UNIVERSITI TEKNOLOGI MALAYSIA

setting up...

Step 1: add dependencies in pom.xml :

```
<!-- Hibernate Core -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.6.5.Final</version> <!-- Use the late
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.2.25.RELEASE</version> <!-- Use the l
</dependency>
<!-- Hibernate EntityManager -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>5.5.6.Final</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.23</version>
</dependency>
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

cont...

Step 2: Hibernate Configuration (using programmatic approach)

```
@Configuration
@EnableWebMvc
@EnableTransactionManagement
@ComponentScan(basePackages = {"config", "controller", "service", "entity", "util"})
public class HibernateConfig implements WebMvcConfigurer {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://localhost:3306/springdb");
        dataSource.setUsername("root");
        dataSource.setPassword("");
        return dataSource;
    }
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

cont...

Step 2: Hibernate Configuration (using programmatic approach)

```
    @Bean
    public LocalSessionFactoryBean sessionFactory(DataSource dataSource) {
        LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();
        sessionFactoryBean.setDataSource(dataSource);
        sessionFactoryBean.setPackagesToScan("entity");

        Properties hibernateProperties = new Properties();
        hibernateProperties.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");
        // you can add more Hibernate properties here as needed
        sessionFactoryBean.setHibernateProperties(hibernateProperties);
        return sessionFactoryBean;
    }

    @Bean
    public HibernateTransactionManager transactionManager(LocalSessionFactoryBean sessionFactory) {
        HibernateTransactionManager transactionManager = new HibernateTransactionManager();
        transactionManager.setSessionFactory(sessionFactory.getObject());
        return transactionManager;
    }
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

cont...

Step 3: Create the Entity Classes. Define relationships if necessary (e.g., @OneToMany, @ManyToOne etc...)

```
@Entity
@Table(name = "customer")
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;
    @Column(name = "name")
    private String name;
    @Column(name = "address")
    private String address;
    @Column(name = "contactNum")
    private String contactNum;
    @Column(name = "email")
    private String email;
    // Constructors, getters, setters, and other methods as always
    // removed for simplicity. please re-generate
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

cont...

Step 4: Implement the Data Access Object (DAO) with Hibernate

```
@Service
public class CustomerDao_usingHibernate {

    @Autowired   //spring dependency injection
    private SessionFactory sessionFactory;

    public void add(Customer customer) {
        Session currentSession = sessionFactory.getCurrentSession();
        currentSession.save(customer);
    }

     public Customer getById(int id) {
        Session currentSession = sessionFactory.getCurrentSession();
        return currentSession.get(Customer.class, id);
    }

    public List<Customer> getAll() {
        Session currentSession = sessionFactory.getCurrentSession();
        return currentSession.createQuery("from Customer").getResultList();
    }
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

cont...

```
@Transactional
public void update(int id, Customer customer) {
    Session currentSession = sessionFactory.getCurrentSession();

    // Retrieve the persistent customer from the database using the provided id
    Customer existingCustomer = currentSession.get(Customer.class, (long) id);

    // Check if the customer exists before updating
    if (existingCustomer != null) {
        // Update the properties of the existing customer with the new values
        existingCustomer.setName(customer.getName());
        existingCustomer.setAddress(customer.getAddress());
        existingCustomer.setContactNum(customer.getContactNum());
        existingCustomer.setEmail(customer.getEmail());
        // Save the changes back to the database
        currentSession.merge(existingCustomer);
    }
}
}
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

cont...

```
    public void delete(int id) {
    Session currentSession = sessionFactory.getCurrentSession();

    // Retrieve the persistent customer from the database using the provided id
    Customer customerToDelete = currentSession.get(Customer.class, (int) id);

    // Check if the customer exists before deleting
    if (customerToDelete != null) {
        // Delete the customer from the database
        currentSession.delete(customerToDelete);
    }
}
```


UTM
UNIVERSITI TEKNOLOGI MALAYSIA

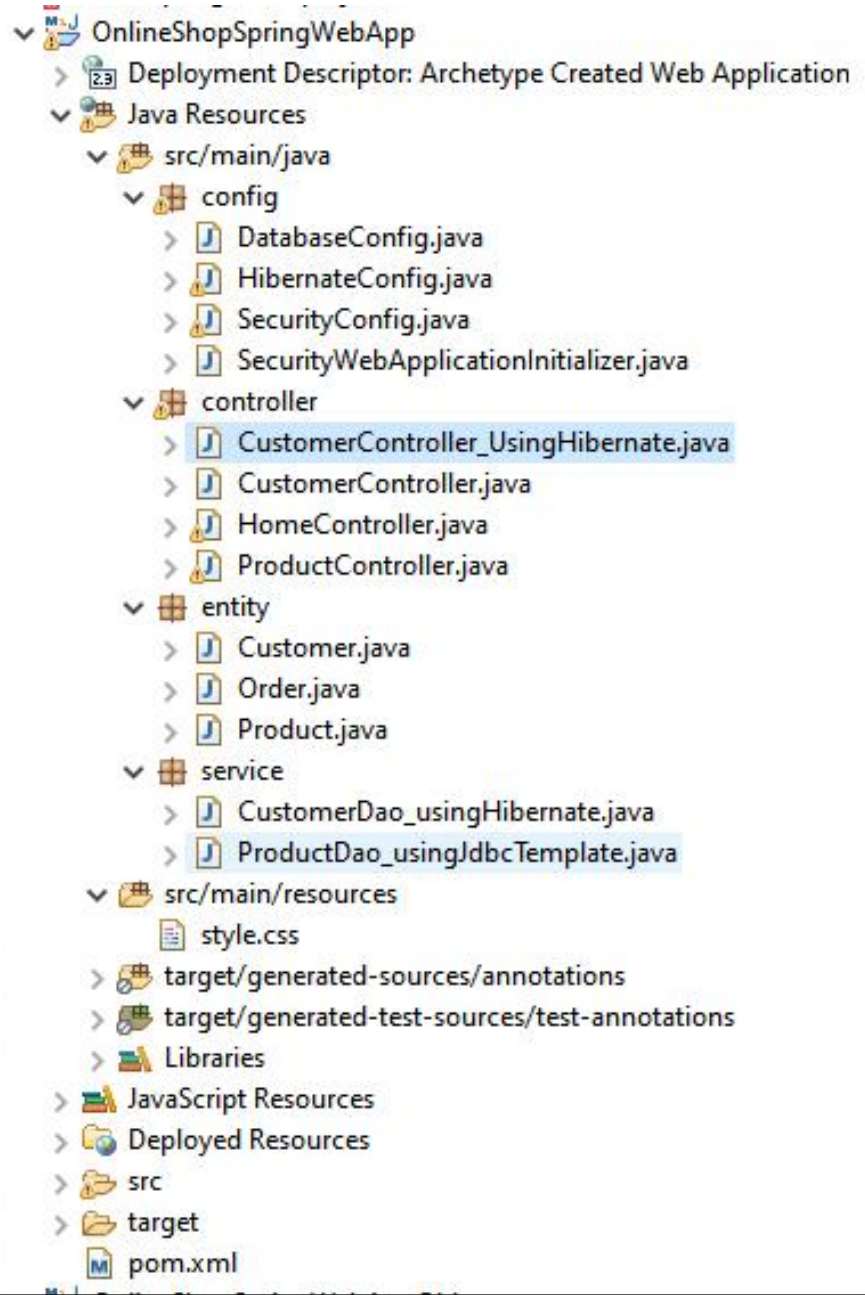
cont...

Step 5: Invoke by the Controller

```
@Controller
@RequestMapping("/usinghibernate")
public class CustomerController_UsingHibernate {

    @Autowired  //spring dependency injection
    private CustomerDao_usingHibernate cDao_usingHibernate;

    @RequestMapping("/getall")
    @ResponseBody()
    public String getall() {
        List<Customer> clist = cDao_usingHibernate.getAll();
        return "this is getall" + clist.toString();
    }

    @RequestMapping("/add")
    @ResponseBody()
    public String add(@ModelAttribute("customer") Customer customer) {
        cDao_usingHibernate.save(customer);
        return "added .. " + customer.toString();
    }
//implement getById, deleteById, updateById here
```


UTM UNIVERSITI TEKNOLOGI MALAYSIA

project structure

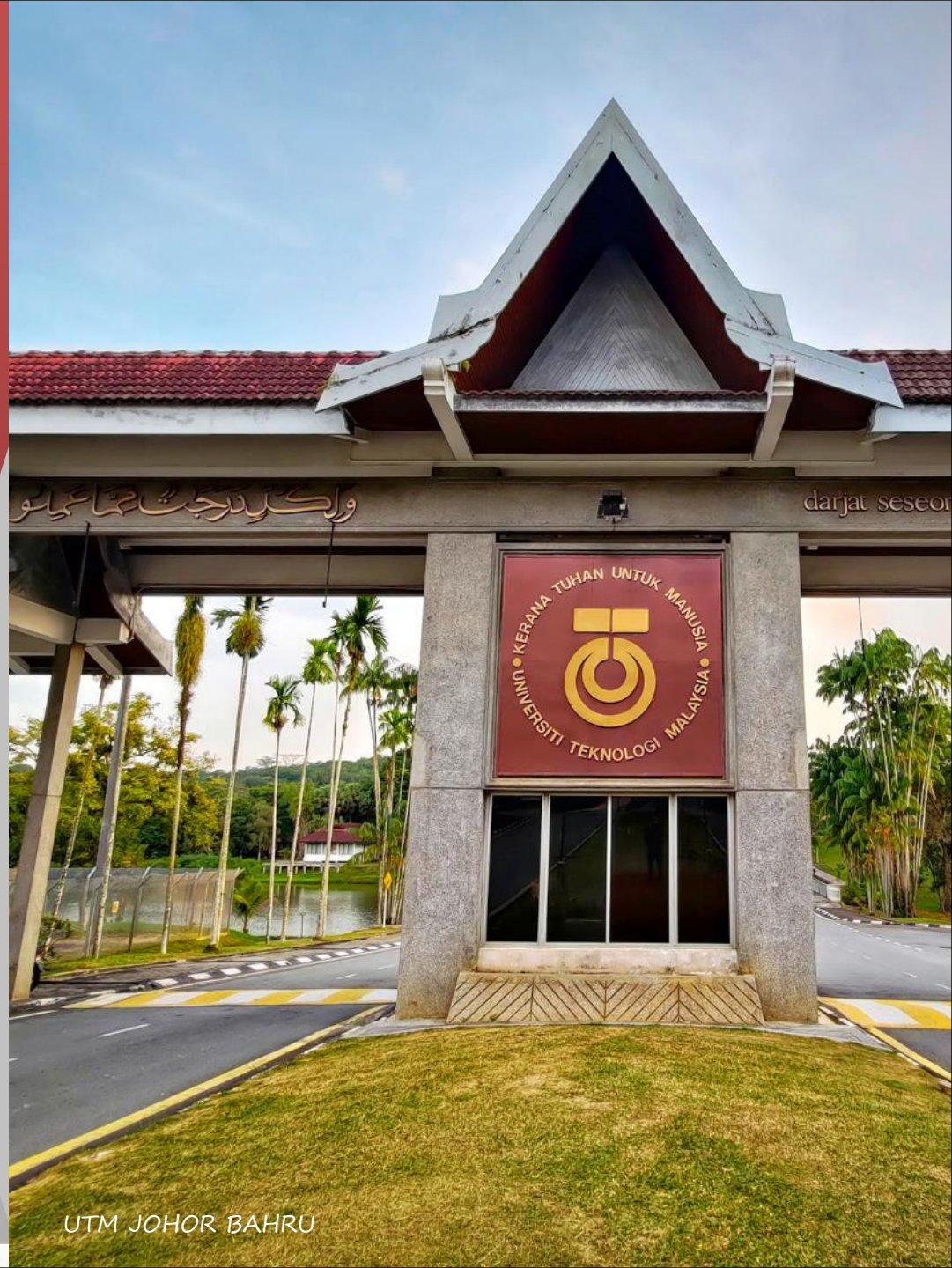


UTM
UNIVERSITI TEKNOLOGI MALAYSIA

TOPIC 8 – Using Relational Database (MySQL)

The End

innovative • entrepreneurial • global



UTM JOHOR BAHRU

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SECJ 3303 – INTERNET PROGRAMMING

TOPIC 9 – EL & JSTL

innovative • entrepreneurial • global

UTM JOHOR BAHRU

TOPIC 9

JSP EL & JSTL
(Expression Language & JSP Standard Tag Library)

- EL (Expression Language) and JSTL (JSP Standard Tag Library) were introduced with the JSP 2.0 specification.
- These tags have several advantages over the older JSP tags that were used prior to the JSP 2.0 specification
- These provides a compact syntax that lets us to get data from JavaBeans, maps, arrays, and lists that have been stored as attributes of a web application
- These tags provide a way to reduce the amount of scripting in your applications. .
- EL + JSTL can improve your JSPs code significantly
- (For most applications, you can use JSTL and EL to remove all JSP scripting ie scriptlet, expression, declaration etc...)
- As a matter of principle – using MVC, a JSP page should be a view page without scriptlets.

JSP EL (Expression Language)

Advantages:

- EL is more compact and elegant. This makes it easier to code and read
- EL makes it easy to access nested properties.
  - simpler syntax ie ${varname} to print value of a simple variable
  - ${user.address.postcode}, ${user.address.state} to print nested properties

- EL lets you access collections such as arrays, maps, and lists easily
- EL handles null values better than standard JSP tags
- EL provides functionality that isn't available from the standard JSP tags.
  - ie, it lets you work with HTTP headers, cookies, and context initialization parameters. It also lets you perform calculations and comparisons

Using EL for Accessing Application Data

TOPIC 9

- EL expressions must be enclosed between **${** and **}**.
- ${data} – scoped variable data.
- The dot (.) operator.
- The bracket ['name'] operator.
- E.g you can you either of these styles.
  - ${user.name}
  - ${user["name"]}

* ${customer.name} is equivalent to ${customer["name"]} is equivalent to ${customer['name']}

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Syntax comparison – Standard JSP Tags vs EL

TOPIC 9

Syntax – Standard JSP tags

```
<jsp:useBean id="user" scope="session" class="model.User"/>
<label>Email:</label>
<span><jsp:getProperty name="user" property="email"/></span><br>
<label>First Name:</label>
<span><jsp:getProperty name="user" property="firstName"/></span><br>
<label>Last Name:</label>
<span><jsp:getProperty name="user" property="lastName"/></span><br>
```

Syntax – EL

```
<label>Email:</label>   <span>${user.email}</span><br>
<label>First Name:</label> <span>${user.firstName}</span><br>
<label>Last Name:</label>  <span> ${user.lastName} </span><br>
```

Examples

code in controller or servlet

```
String name = “muhammad”;
String nameArr[] = {“ali”, “siti” ,”zaki”};
Person p = new Person(“dina”,”female”,23);
request.setAttribute(“name”,name);
request.setAttribute(“nameArr”,nameArr);
request.setAttribute(“p”,p);
```

code in JSP page

```
Hello ${name}                    //prints Hello muhammad
Hello ${nameArr[0]}              //prints Hello ali
Hello ${nameArr[2]}              //prints Hello zaki
Helllo ${p.gender}
or Hello ${p[“gender”]}          //prints Hello female
```

Operators in EL

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/ (or div)	Division
% (or mod)	Modulus (Reminder)
== (or eq)	Equality
!= (or ne)	Inequity
< (or lt)	Less than
> (or gt)	Greater than
<= (or le)	Less than or equal
>= (or ge)	Greater than or equal
&& (or and)	Logical AND
|| (or or)	Logical OR
! (or not)	Boolean complement
empty	Check for empty value

Other Operators

Syntax	Description
`empty x`	Returns true if the value of x is null or equal to an empty string.
`x ? y : z`	If x evaluates to true, returns y. Otherwise, returns z.

Example	Result
`${empty firstName}`	`true if firstName returns a null value or an empty string`
`${true ? "s1" : "s2"}`	`s1`
`${false ? "s1" : "s2"}`	`s2`

Examples

Code in jsp page

```
<h4>1+3 result is : ${1+3 }</h4>
<h4>7/2 result is : ${7/3 }</h4>
<h4>9 % 2 result is : ${9 % 2}</h4>
<h4>9 gt 5 result is : ${9 gt 5 }</h4>
<h4>9 le 5 result is : ${9 le 5 }</h4>
<h4>9 ne 5 result is : ${9 ne 5 }</h4>
<h4>9 < 5 result is : ${9 < 5 }</h4>
<h4>Result is : ${9 > 5 }</h4>
<h4>9 gt 5? : ${9 gt 5? "yes it is true": "no it is false" }</h4>
```

Result :

```
1+3       result is : 4
7/2       result is : 2.3335
9 % 2     result is : 1
9 gt 5    result is : true
9 le 5    result is : false
9 ne 5    result is : true
9 < 5     result is : false
9 > 5     result is : true
9 gt 5? : yes it is true
```

Implicit Objects in EL

Implicit Object	Content
pageScope	access to the scoped variables
requestScope	access to the scoped variables
sessionScope	access to the scoped variables
applicationScope	access to the scoped variables
param	a Map object. `param["foo"]` returns the first string value associated with request parameter *foo*.
paramValues	a Map object. `paramValues["foo"]` returns an array of strings associated with request parameter *foo*.
header	a Map object. `header["foo"]` returns the first string value associated with header *foo*.
headerValues	a Map object. `headerValues["foo"]` returns an array of strings associated with header *foo*.
initParam	access to context initialization parameters
cookie	exposes cookies received in the request
pageContext	`PageContext` properties (e.g. `HttpServletRequest`, `ServletContext`, `HttpSession`)

JSTL

- JSTL provides tags for common tasks that need to be performed in JSPs
- JSTL + EL can be used to replace scriplet code from a JSP page
- To use JSTL tags, we must make the jstl-impl.jar and jstl-api.jar files available to the application (add into lib, or add depencency if use maven)
- also need to add the taglib directive that identifies the JSTL library and its prefix

The 5 JSTL Libraries

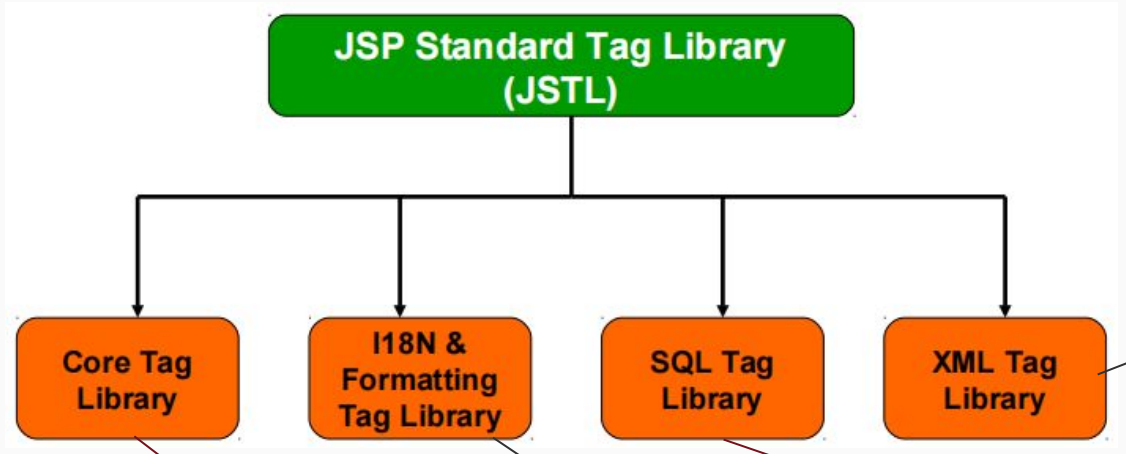
JSP Standard Tag Library (JSTL) provides tags for common tasks that need to be performed in JSP.
The 5 JSTL Libraries are :-
(i) **core library:** This library contains tags that you can use to encode URLs, loop through collections, and code if/else statements.
If you use the MVC pattern, the tags in **the core library are often the only JSTL tags you'll need** as you develop your JSPs.

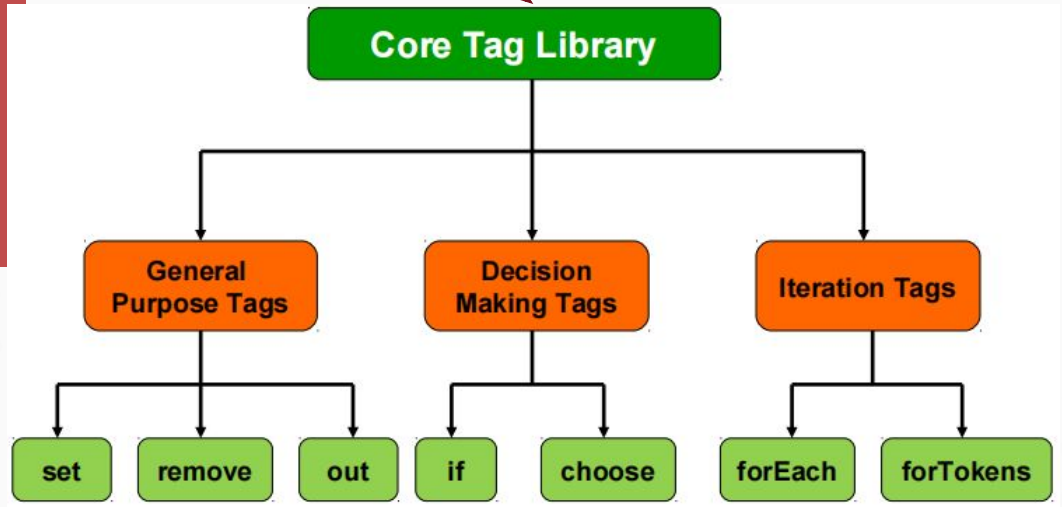
If necessary, though, you can use the other four libraries to work with (ii) internationalization, (iii) databases, (iv) XML, and (v) strings

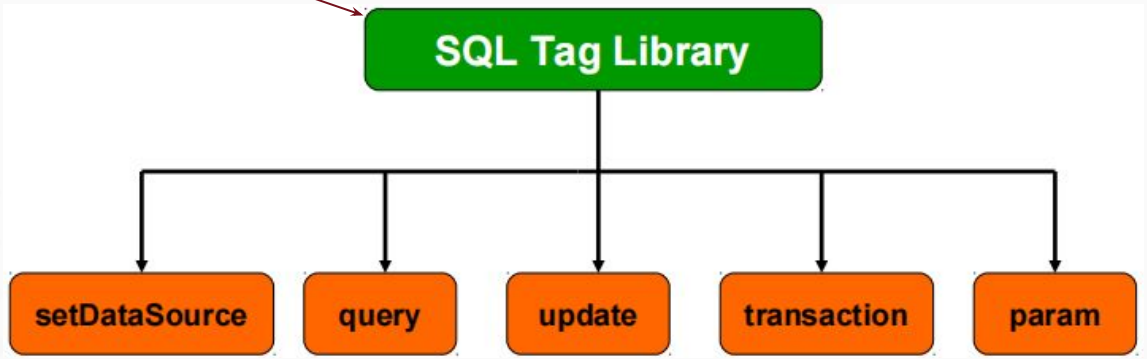
In this course, will only cover the (i) JSTL Core Library/ i.e c:out, c:forEach, c:if

The primary JSTL libraries

Name	Prefix	URI	Description
Core	`c`	http://java.sun.com/jsp/jstl/core	Contains the core tags for common tasks such as looping and if/else statements.
Formatting	`fmt`	http://java.sun.com/jsp/jstl/fmt	Provides tags for formatting numbers, times, and dates so they work correctly with internationalization (i18n).
SQL	`sql`	http://java.sun.com/jsp/jstl/sql	Provides tags for working with SQL queries and data sources.
XML	`x`	http://java.sun.com/jsp/jstl/xml	Provides tags for manipulating XML documents.
Functions	`fn`	http://java.sun.com/jsp/jstl/functions	Provides functions that can be used to manipulate strings.


JSP Standard Tag Library (JSTL)
Core Tag Library
I18N & Formatting Tag Library
SQL Tag Library
XML Tag Library


Core Tag Library
General Purpose Tags
Decision Making Tags
Iteration Tags
set
remove
out
if
choose
forEach
forTokens


SQL Tag Library
setDataSource
query
update
transaction
param

JSTL core

- JSTL Core provide support for output, iteration, conditional logic, catch exception, url, forward or redirect, response etc.
- Before you can use JSTL tags within an application, you must make the **(i) jstl-impl.jar** and **(ii) jstl-api.jar** files available to the application (**add jstl-1.2.jar to your project lib folder**, or add dependency for maven project)
- we also need to include the taglib in the JSP page as below:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

Core Tags Library

Tag	Descriptor
<c:when>	Sub tags of <c:choose> that includes its body if its condition evaluates to 'true'.
<c:otherwise>	Sub tags of <c:choose> that includes its body if its condition evaluates to 'false'.
<c:forEach>	for iteration over a collection
<c:forTokens>	for iteration over tokens separated by a delimiter.
<c:param>	used with <c:import> to pass parameters
<c:url>	to create a URL with optional query string parameters

JSTL Core Tags

Tags	Description
<c:out>	To write something in JSP page, we can use EL also with this tag
<c:import>	Same as <jsp:include> or include directive
<c:redirect>	redirect request to another resource
<c:set>	To set the variable value in given scope.
<c:remove>	To remove the variable from given scope
<c:catch>	To catch the exception and wrap it into an object.
<c:if>	Simple conditional logic, used with EL and we can use it to process the exception from <c:catch>
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <c:when> and <c:otherwise>

Using the general purpose – c:out tag

TOPIC 9

code in controller or servlet

```
String name = “muhammad”;
String nameArr[] = {“ali”, “siti” ,”zaki”};
Person p = new Person(“dina”,”female”,23);
request.setAttribute(“name”,name);
request.setAttribute(“nameArr”,nameArr);
request.setAttribute(“p”,p);
```

code in JSP page

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
:
<c:out value="Hello ${name}" /><br>                  //prints Hello muhammad
<c:out value="Hello ${nameArr[0]}" /><br>            //prints Hello ali
<c:out value="Hello ${nameArr[2]}" /><br>            //prints Hello zaki
<c:out value="You are a ${p.gender}"  /><br>         //prints You are a female
or Hello ${p["gender"]}          /><br>
```

Using the iterator – c:forEach tag

- You can use the forEach tag to loop through items that are stored in most collections (i.e array, list, etc..)

- You can use the var attribute to specify the variable name that is used to access each item within the collection.
- You can use the items attribute to specify the collection that stores the data.
- If necessary, you can nest one forEach tag within another.

Using the iterator – c:forEach tag

code in controller or servlet

```
String nameArr[] = {"ali", "siti" ,"zaki"};
request.setAttribute("nameArr",nameArr);
List<Person> pList = new ArrayList<Person>();
pList.add(new Person("dina","female",23));
pList.add(new Person("kali","male",32));
pList.add(new Person("Saly","female",17));
request.setAttribute("pList",pList);
```

code in JSP page

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
:
to print pList
<c:forEach var="person" items="${pList}" >
        Name  : ${person.name} <br>
        Gender : ${person.gender}<br>
        Age   : ${person.gender}<br><br>
</c:forEach>
```

cont ...

TOPIC 9

code in JSP page

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
:
to print array name
<c:forEach var=”name” items=”${nameArr}” >
      Name  : ${name} <br>
</c:forEach>


//prints       Name ali
                Name siti
                Name zaki
```

Using the decision making- c:if tag

TOPIC 9

code in controller or servlet

```
Person p = new Person("dina","female",23);
request.setAttribute("p",p);
```

code in JSP page

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
:
<c:if test="${p.gender == 'female'}" >
    <c:out value="you can wear a pink shirt tomorrow" />
</c:if>
<c:if test="${p.gender == 'male'}" >
    <c:out value="lets play futsal this evening" />
</c:if>
```

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

TOPIC 9 – JSP EL & JSTL

the end

innovative • entrepreneurial • global

UTM JOHOR BAHRU